Projekt 0.6 RSA kryptering

(Bagest i projektet er der svar til de øvelser, der er gulet)

1. Introduktion. Nøgler til kryptering

Alle former for kryptografi prøver at løse følgende problem: En *afsender, A* ønsker at sende en meddelelse til en *modtager, M*, således at den ikke kan læses af uvedkommende. Afsenderen koder derfor meddelelsen ved brug af en "nøgle". Modtageren afkoder den kodede meddelelse ligeledes ved brug af en "nøgle".

Derfor er et godt billede på, hvad der foregår, at afsenderen *låser sin meddelelse inde i en boks med brug af sin A-nøgle. Modtageren låser boksen op med brug af sin M-nøgle*. Andre kunne finde meddelelsen, hvis de kunne lave sig en M-nøgle, eller hvis de på anden vis kunne dirke låsen op.

I klassiske systemer har A-nøgler og M-nøgler været (stort set) identiske. Man har tænkt i baner som: En nøgle kan både låse i og låse op. Men det er jo ikke rigtigt.
Tænk på en banks døgnboks, hvori en forretning lægger dagens omsætning, når banken er lukket. "Afsenderen" har her en nøgle, der ikke er spor hemmelig; blot med et beskedent gebyr kan enhver blive tilkoblet denne døgnboks-service og få sig en nøgle. Dette er altså en offentlig nøgle.
Med nøglen åbnes boksen, kuverten med pengene lægges i, og boksen lukkes. Hvis man nu fortryder, er det for sent: Luk op igen og se - kuverten er forsvundet. Naturligvis, for ellers ville den næste jo kunne tage pengene. Du skal have en helt anden hemmelig nøgle, for at kunne komme ind til de deponerede penge.

Enhver kender i øvrigt dette, at man kan låse noget inde, uden at kunne låse op igen. Måske har man prøvet at smække sig ude! Eller hvis man har en cykel, eller et rum, der er låst med en smæklås (feks en hængelås), ja så kan man jo låse, uanset om man har nøglen eller ej. Det er denne tankegang moderne kryptografiske metoder bygger på. Lad blot den "nøgle", som afsenderen skal bruge til at kode sin meddelelse med, være offentligt tilgængelig, og sørg alene for, at nøglen til at afkode med er hemmelig. Det kræver selvfølgelig, at de to nøgler er vidt forskellige, og at den ene ikke kan findes ud fra kendskab til den anden.

Men man skal også sikre sig mod indbrud, feks af hackere. Derfor er det et krav, at når meddelelsen er sendt afsted i kodet form, er det næsten umuligt for uvedkommende at nå ind til den. I matematisk talteori svarer dette til begrebet en *énvejsfunktion*. Dette er en funktion E(x) fra en talmængde til en anden, hvor der gælder:
- $E(x)$ er altid let at beregne (dvs det er let at kode)
- Hvis man kender $E(x)$ er det et beregningsmæssigt svært (i praksis umuligt) problem at finde x, altså at svare på, "hvor $E(x)$ kommer fra".

Eksempel 1

Lad os sige vi har givet tallene $a = 123$, $b = 3589$ og $y = 1241$. De er offentligt kendte.

Find nu et tal x, således at 123^x ved division med 3589 får en rest på 1241. Det er et meget vanskeligt problem. Vælges a, b og y betragteligt større, feks tal med 30, 40 eller 100 cifre, så er det praktisk taget umuligt at løse. Dvs den funktion, der til et x knytter den rest, vi får ved at dividere 3589 op i 123^x er en envejsfunktion. Den slags problemer behandles i *talteori*

Envejsfunktioner vil sikkert finde vid udbredelse som beskyttelse mod feks hackere. Er du ulovligt på vej ind i et system, så er den første elementære beskyttelse, du skal bryde igennem ofte at opgive korrekt brugernummer og password. Det bremser de fleste. Men dygtige hackere ved jo, at disse tal findes inde i maskinens lager, da den skal checke de oplysninger, vi giver den. Og af og til kan de komme "bagom" og finde disse tal. Det mest sikre er derfor, at tallene slet ikke findes i lageret! Hvordan det? Jo, i stedet findes en envejsfunktion $E(x)$, samt de tilladte y-værdier. Du skal opgive dine x-værdier, maskinen beregner $E(x)$ og sammenligner med de korrekte y-værdier. Og selv om en hacker/kodebryder finder y-værdierne, kan vedkommende ikke regne sig tilbage til x-værdierne og skaffe sig adgang.

Nu er det en dårlig fidus, hvis ingen overhovedet kunne "låse sig ind til" meddelelsen. Det er en speciel type envejsfunktion vi er interesseret i indenfor kryptologi. Nemlig dem, hvorom der gælder, at nogen kan lukke dem op. I talteorien kaldes sådanne funktioner malende for *smæklåsfunktioner*.
En smæklåsfunktion er en envejsfunktion $E(x)$, hvorom der gælder, at givet en bestemt ekstra information er det en let sag at finde x ud fra $E(x)$.
Det er en let sag at låse noget inde og smække låsen. Men det er svært at lukke op igen. Er låsen meget kompliceret, kan det være en næsten umulig opgave at "dirke sig ind" - men med nøglen i hånden er det selvsagt let.

Hele arbejdet i det følgende drejer sig om at nå frem til at forstå, hvordan vi kan lave sådanne envejs- og smæklåsfunktioner, der udgør eller kommer til at udgøre hjertet i de nye sikkerhedssystemer inden for elektronisk kommunikation, feks ved brug af Dankort, netbank osv. En af disse metoder bygger på talteori, nærmere bestemt på (vanskeligheden i at) faktorisere store tal i primfaktorer.

Den talteori, der indtil for få år siden blev opfattet som helt ren matematik, giver os værktøjet til den mest effektive kryptering, der tænkes kan. Det som matematikerne troede var milevidt fra al anvendt matematik, er pludselig blevet et kraftigt voksende kommercielt område. Matematikere kan nu leve af at producere primtal!

Det system vi kalder RSA-systemet er opkaldt efter de tre matematikere Rivest, Shamir og Adleman, der opfandt og beskrev det første gang i 1977. Det er et eksempel på et såkaldt *Public Key System*. Den grundlæggende del af matematikken har været kendt i mange hundrede år, noget siden oldtiden. Men en masse sofistikerede metoder og opdagelser fra nyere tid har muliggjort, at systemet kan anvendes til sikker datakommunikation i dag.

2. Forudsætninger

Hvis dette projekt gennemføres i forlængelse af projekterne 0.4 og 0.5, så kan du springe dette afsnit over. Her vil vi i koncentreret form præsentere nogle af de værktøjer og begreber vi anvender, når vi præsenterer RSA algoritmen og viser, hvorfor den virker.

2.1 Primtal og faktorisering

Aritmetikkens fundamentalsætning)
Ethvert helt tal kan skrives på en og kun en måde som et produkt af primtal, dvs *primtalsfaktoriseringen af et helt tal er entydig*.

Vi kalder denne opskrivning for *primfaktoropløsningen af N*. Sætningen siger altså at primfaktoropløsningen er éntydig. Beviset gives i projekt 0.5.

Eksempel 2. Primfaktoropløsninger

a) $2310 = 2 \cdot 3 \cdot 5 \cdot 7 \cdot 11$ b) $574821 = 3^2 \cdot 13 \cdot 17^3$

Øvelse 1
Opskriv uden brug af værktøj en primfaktoropløsning af :

a) 42	b) 81	c) 225	d) 117
e) 1368	f) 2093	g) 1024	h)1025

Alle værktøjsprogrammer kan primtalsfaktorisere. I TI Nspire gøres med funktionen *factor()*. I Maple gøres det med funktionen *ifactor()*. Undersøg hvordan dit værktøjsprogram fungerer.

Øvelse 2
Anvend dit værktøj til at opskrive en primfaktoropløsning af:

a) 3397 b) 1991009 c) 560560 d) $2^{32}+1$ e) $2^{100}+1$

Eksempel 3. Primtalsfaktorisering af store tal
Det er en enkel sag at gange to primtal *p* og *q* og få et resultat $N = p \cdot q$. Selvom *p* og *q* er meget store, kunne vi klare det i hånden. Men det er en vanskelig sag at finde ud af, hvilke primtal der går op i et givet stort tal.
Selv for moderate tal er det svært uden værktøj. *p* = 1409 og *q* = 1993 er primtal. Deres produkt er *N* = 2.808.137. Var vi startet med *N*, ville det være ret svært at finde *p* og *q*. Prøv feks at finde hvilke primfaktorer der er i 3.059.899, uden brug af værktøj.

Hvis det var tal 100 eller 200 cifre var opgaven praktisk taget umulig at løse i hånden.

Eksempel 4. Eratosthenes si
En af oldtidens største videnskabsmænd var Eratosthenes fra Kyene, der levede ca. 284-202 fvt. Han er mest berømt for at have beregnet Jordens omkreds med stor nøjagtighed, men han beskæftigede sig også med talteori og beskrev en metode til at afgøre om et givet tal er et primtal. Eratosthenes vidste, at der er uendeligt mange primtal – et bevis herfor findes i Euklids *Elementer*, og er gengivet i projekt 0.5 – og hans metode er simpelthen at gå frem i talrækken og skridt for skridt slette alle tal i 2-tavellen, i tre-tabellen, i 5-tabellen, i 7-tabellen osv., dvs. i alle primtalstabeller. Når vi kommer t il det givne tal *N* kan vi blot tjekke om det er blevet slettet. Hvis ikke er det et primtal. En anden måde at beskrive denne *Eratosthenes si*, der kun lader primtallene sive igennem, er at stille primtallene op i række, og spørge: Går 2 op i *N*? Går 3 op i *N*? Går 5 op i *N*? Går 7 op i *N*? … Går p_i op i *N*?, hvor p_i er det i'te primtal, osv.

Øvelse 3.
a) Hvor langt op i talrækken behøver vi at fortsætte vores undersøgelse for at se om fx 1234567891 er et primtal?

Der findes en besynderlig formel, der giver et tilnærmet tal til antallet af primtal mindre end et givet tal *n*. Allerede Gauss opdagede som barn denne formel ved at læse i primtalstabeller!
Formlen siger, at antallet af primtal mindre end *n*, der betegnes med $\pi(n)$ kan estimeres med:

$\pi(n) \approx \frac{n}{\ln(n)}$, hvor ln er den naturlige logaritme.

Dvs antal primtal mindre end 1 million er af størrelsesorden:

$$\pi(1000000) = \pi(10^6) \approx \frac{10^6}{\ln(10^6)} = 7382$$

Øvelse 4
Hvor mange primtal findes der omtrentlig, der har mindre end 100 cifre.

Øvelse 5.
En definition på titalslogaritmen log er, at givet et tal *N*, da er log(*N*) det tal *a*, der opfylder: $10^a = N$
Logaritmen til tallet 100 er derfor 2, og logaritmen til tallet 1 million er 6. Logaritmen til 5 milliarder er ca 9.7.
Vi lægger mærke til, at antallet af cifre i et tal (i 10-talssystemet) derfor kan findes ved at udregne (10-tals-)logaritmen til tallet, og så forhøje til nærmeste hele tal.
a) Hvor mange cifre er der i tallet $2^{50}+1$
b) For hvilket tal *a* har tallet 2^a+1 100 cifre

Øvelse 6.
De tal, man arbejder med i moderne kryptering har flere hundrede cifre. Sådanne tal kan vores værktøjsprogrammer ikke klare. Men 50-cifrede og selv 100-cifrede tal klarer de, måske med en vis ventetid.
a) Hvor mange cifre har tallet $2^{200}+1$?
Anvend dit værktøj til at faktorisere tallet.
b) Hvor mange cifre har tallet $2^{330}+1$?
Anvend dit værktøj til at faktorisere tallet.
c) Faktoriser tallet $10^{100}+1$.
Bemærkning. Hvis du vil udfordre dit værktøj med meget større tal, så sørg for, at du har gemt det, som du i øvrigt arbejder med i værktøjsprogrammet, da du kan sætte værktøjet skakmat, så du ikke kan komme videre.

Vi lægger mærke til, at i faktoriseringen optræder normalt en del mindre primtal. Hvis programmet arbejder med nogle første forsøg, hvor der anvendes en algoritme som Eratosthenes si, så vil det oprindelige tal hurtigt blive faktoriseret ned til noget mere overkommeligt.

I RSA kryptografi bygger man på, at det offentligt kendte nøgletal i praksis ikke kan faktoriseres, fordi det er bygget op ved hjælp af to meget store primtal. I et forsøg på at knække en kode, dvs primtalsfaktorisere nøgletallet, vil man ikke kunne reducerer problemet ved hurtigt at faktorisere en række mindre primtal ud. Der er kun to primtal, og de er hver for sig meget store. Men heri ligger også en af udfordringerne for dem, der konstruerer koderne. For hvordan ved de med sikkerhed, at de tal, de bygger koden op med, faktisk er primtal? Det ved de h eller ikke nødvendigvis, men de ved at sandsynligheden for det er meget stor. Et af de store felter inden for denne del af matematikken er de såkaldte *primtalstest*, hvor det første og ret simple test knytter sig til *Fermats lille sætning*, der er omtalt og bevist i projekt 0.4

Fermats lille sætning
1) Hvis *p* er et primtal, og *a* er et tal, som *p* ikke går op i, så gælder der: $a^{p-1} \equiv 1 \pmod{p}$
2) Hvis *p* er et primtal, og *a* er vilkårligt tal, så gælder der: $a^p \equiv a \pmod{p}$

Bemærkning. Moduloregning, dvs regning hvor vi kun interesserer is for resten ved division med et tal, er omtalt nedenfor.

De matematiske værktøjsprogrammer kan gennemføre modulo-regning. Hvis vi ville tjekke Fermats lille sætning på $p=17$ kunne det i maple se sådan ud:

$$seq\left(a^{16} \bmod 17,\ a=1..16\right) \xrightarrow{\text{to list}} [1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1]$$

Det betyder, at for alle tal a^{16}, fx 5^{16} får vi resten 1 ved division med 17.
Version 2 af Fermats lille sætning ville se således ud:

$$seq\left(a^{17} \bmod 17,\ a=1..16\right) = 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 9,\ 10,\ 11,\ 12,\ 13,\ 14,\ 15,\ 16$$

Øvelse 7. Fermats lille sætning undersøgt med værktøjsprogram
Anvend dit værktøjsprogram til at undersøge Fermats lille sætning med primtallet 23 og det sammensatte tal 27.

Øvelse 8. Carmichael tal
Fermats lille sætning er som nævnt det første simple primtalstest, dvs et test på, om et tal er et primtal. Hvis *p* er et primtal, skal Fermats lille sætning gælde. Vi kan desværre ikke slutte modsat, da der findes tal *p*, som opfylder Fermats lille sætning, uden at være primtal. Disse tal kaldes pseudo-primtal, eller Carmichael tal. Der er ikke mange af dem, men de findes. Det eneste Carmichael tal under 1000 er tallet 561.
Prøv at udregne $a^{561} \pmod{561}$ for alle tal fra 1 til 560 med brug af dit værktøjsprogram, fx ved at udregne:

$$seq\left(a^{561} \bmod 561,\ a=1..560\right)$$

Eksempel 5. Fermat-tal
Fermat, der var jurist af profession, var en stor matematisk begavelse. Han medvirkede i udviklingen af differentialregningen, i sandsynlighedsregningens undfangelse og meget andet - og altså også i talteorien. Han er mest berømt for det, der i matematikhistorien er blevet kaldt *Fermats store sætning* eller *sidste* sætning, der siger, at hvis n er <u>større</u> end 2, så findes ingen hele tal x,y og z, der opfylder ligningen:

$$x^n + y^n = z^n$$

(for n = 2 er det jo Pythagoras)
Sætningen blev bevist efter 350 år af Andrew Wiles i 1996 (?).
Fermat har givet flere bidrag til talteorien. Han mente at tallene der kunne skrives på formen: $F_n = 2^{2^n}+1$ alle er primtal. Tallene er siden blevet kaldt Fermat-tallene og betegnes F_n. Påstanden er sand for n = 1,2,3,4, hvor Fermat-tallene er:

$F_1 = 5$, $F_2 = 17$, $F_3 = 257$, $F_4 = 65.537$

men Euler viste desværre for Fermat, at der om det næste tal gælder:

$$F_5 = 4294967297 = 641 \cdot 6700417$$

så F_5 er ikke et primtal. I dag tror man snarere at ingen af de følgende Fermat-tal er primtal.

2.2. Division med rest og primiske tal

Praxis. Division med rest
Mængden af hele tal (positive, negative og nul) betegnes $\mathbb{Z}$. At et tal a er et helt tal angives med: $a \in \mathbb{Z}$, der læses *a tilhører* $\mathbb{Z}$.

Når vi har to vilkårlige hele tal, $a,b\in\mathbb{Z}$ kan vi dividere a op i b ved den metode, vi lærte i folkeskolen. Resultatet skrives således:

$b=q\cdot a+r$, hvor $q\in Z$, og $0\le r<a$ (*)

Vi vil altid skrive resultatet således, at resten *r* ligger i dette interval. Denne rest kaldes *den principale rest.*

Opskrivningen af (*) kaldes *divisionsligningen*. Vi viser i projekt 0.4, at divisionsligningen er éntydig.

Hvis *a går op i b*, dvs hvis resten er 0, siger vi at *a er divisor i b*, og vi skriver: $a\,|\,b$.

Eksempel 6

a = 5, *b* = 32: $32=6\cdot5+2$

a = 3, *b* = 16: $16=5\cdot3+1$

a = 3, *b* = -16: $-16=-6\cdot3+2$

Bemærk at kravet om $0\le r<a$ giver en lidt anden divisionsligning for negative tal.

Eksempel 7

$6\,|\,216$ $37\,|\,2.954.524$ $113\nmid965.356$ (”går ikke op i”)

Praxis. Største fælles divisor og indbyrdes primiske tal.

Givet to tal $a,b\in\mathbb{Z}$. Det største tal blandt alle de fælles divisorer i *a* og *b* kaldes *den største fælles divisor i a og b* og betegnes med $sfd(a,b)$, $gcd(a,b)$, eller blot (a,b).

Den største fælles divisor for to hele tal a og b er produktet af deres fælles primfaktorer.

Hvis tallet d er den største fælles divisor af to tal *a* og *b* ($(a,b)=d$), så findes tal *s*, og *t*, så *d* kan skrives som en *linearkombination* af *a* og *b*:

$d=s\cdot a+t\cdot b$, hvor $s,t\in\mathbb{Z}$

Hvis den største fælles divisor for *a* og *b* er 1, kaldes *a* og *b* for *indbyrdes primiske*.

Egenskab med linearkombinationen er meget vigtig. Den anvendes i kryptografien til at bestemme den hemmelige nøgle ud fra den nøgle, vi offentliggør. Det vender vi tilbage til nedenfor i afsnit 3. Egenskaben er bevist i projekt 0.5.

Eksempel 8

(10,25) = 5, og $1\cdot25-2\cdot10=5$

(56,15) = 1, og $-4\cdot56+15\cdot15=1$

Bemærkning. Der findes en metode (en algoritme) til at bestemme *s* og *t*. Denne metode kaldes Euklids algoritme, og er behandlet i projekt 0.5.

Øvelse 9

a) Vis, at største fælles divisor af tallene 1309 og 1235 er tallet 1

Man kan med brug af Euklids algoritme bestemme *s* og *t* så

$1=s\cdot1309+t\cdot1235$

De fleste værktøjsprogrammer har indbygget Euklids algoritme. I Maple kan man skrive således:

$igcdex(tal1,\ tal2,'s','t'),\ s,\ t$

hvorefter man dels får oplyst, at 1 er største fælles divisor, og dels får oplyst de to tal *s* og *t*.

b) Bestem to tal *s* og *t* så:

$$1 = s \cdot 1309 + t \cdot 1235$$

Moduloregning og restklasser

Vi anvender moduloregning og restklasser mange gange om dagen, nemlig når vi taler om tid, om hvad klokken er, om hvor lang tid der er til et eller andet, vi har aftalt, dvs. når vi udmåler tid med et ur. Når tiden udmåles i timer, regner vi modulo 24 (eller 12), og når tiden udmåles i minutter regner vi modulo 60. Vi siger ikke, at klokken er 80 minutter over 10, men at den er 20 minmutter over 11. Når klokken passerer midnat, tæller vi ikke videre på tallinjen med 25, 26 osv., men forfra – om natten er klokken 1, 2 osv. Siger vi, at vi går i seng KL 23, regner vi modulo 24, mens de som siger, at de går i seng KL 11, regner modulo 12. Regner man modulo 12, "identificerer" man altså 23 og 11.

Går man i seng KL 23 og sætter uret, så man kan sove i 8 timer, så står man ikke op kl. $23+8=31$, men kl. 7. Tallet 7 får vi matematisk, ved at trække 24 fra 31. I praksis tæller de fleste nok op til 24 (det var én time) og resten af de 8 timer, altså tallet 7 angiver så klokkeslettet, hvor vi står op. Også her "identificerer" vi altså 31 og 7. Men vi kan naturligvis ikke skrive: $31=7$. Derfor har man i matematik indført en særlig betegnelse for denne måde at identificerer tal på, nemlig ved at skrive:

$$31 \,(\mathrm{mod}\, 24) = 7 \,(\mathrm{mod}\, 24)$$

"mod 24" læses *modulo 24*, og angiver, at vi trækker 24 fra tallet lige så mange gange vi kan, indtil vi har et tal mellem 0 og 24. Således gælder altså:

$$48 \,(\mathrm{mod}\, 24) = 0 \,(\mathrm{mod}\, 24) \qquad \text{og} \qquad 245 \,(\mathrm{mod}\, 24) = 5 \,(\mathrm{mod}\, 24)$$

Det sidste udtryk kan vi tolke således: Hvis klokken nu fx er 9, så er den om 245 timer $9+5=14$.
5 kan opfattes som *resten* vi får ved division af 245 med 24. Divisionen går jo ikke op, men giver 10 og altså 5 til rest.
Vi kunne også regne tilbage i tiden:

$$-20 \,(\mathrm{mod}\, 24) = 4 \,(\mathrm{mod}\, 24)$$

Dette kan vi tolke således. Hvis klokken nu fx er 9, så var den for 20 timer siden $9+4=13$.

Tilsvarende gælder der:

$$80 \,(\mathrm{mod}\, 60) = 20 \,(\mathrm{mod}\, 60) \qquad \text{og} \qquad 380 \,(\mathrm{mod}\, 365) = 15 \,(\mathrm{mod}\, 365)$$

Prøv at give en fortolkning af disse to udtryk.

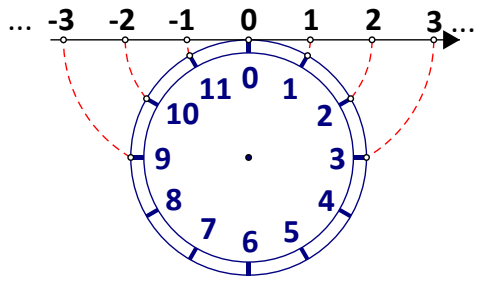
Regner vi modulo 24, så identificerer vi altså alle tallene:

$$\{\ldots, -44, -20, 4, 28, 52, \ldots\}$$

Tilføj selv yderligere to negative og to positive tal.
En sådan mængde af tal kalder vi for en *restklasse modulo 24*. Vi siger også, at tallene i en sådan restklasse er *kongruente modulo 24*, og anvender symbolet $\equiv$ til at udtrykke dette. Vi skriver fx: $4 \equiv 52 \,(\mathrm{mod}\, 24)$.

Øvelse 10

a) Opskriv restklasen hørende til tallet 0, og restklassen hørende til tallet 10.
b) Hvor mange forskellige restklasser modulo 24 findes der?



I almindelighed kan man ved *restklasser modulo n*, hvor *n* er et naturligt tal, forestille sig, at man vikler en tallinje rundt om en cirkel, der har omkredsen *n*. Hver gang vi går *n* positioner frem på den omviklede tallinje rammer vi altså det samme punkt på cirklen.
Restklasserne repræsenteres af tallene {0, 1, 2, …, *n*-1}, der kaldes for *de principale rester* ved division med *n*.
På illustrationen ser man fx, at tallene -3 og 9 er i samme restklasse og altså er kongruente modulo 12.

Eksempel 9

$7 \equiv 12 \pmod 5$ $\quad$ $7 \equiv 122 \pmod 5$

$-3 \equiv 1 \pmod 4$ $\quad$ $-3 \equiv 17 \pmod 4$

Vi skriver ikke altid (mod *n*) efter tallet, hvis dette tal er den principale rest. I stedet tillader vi os for nemheds skyld at skrive eksempelvis $12 \pmod 5 = 2$. Her står, at den principale rest ved division af 12 med 5 er 2.

Øvelse 11
Bestem følgende:
a) $21 \pmod 3$ b) $558 \pmod{17}$ c) $5306509 \pmod{10}$ d) $-20 \pmod 3$ e) $123123123 \pmod 9$

Praxis. Moduloregning
Mængden af principale rester ved division med *n* betegnes: $\mathbb{Z}_n = \{0,1,2,\ldots,n-1\}$
Et tal i $\mathbb{Z}_n$ opfattes som repræsentant for sin tilsvarende restklasse.
Hvis $a,b \in \mathbb{Z}_n$ definerer vi addition af restklasser således:

$$a+b = (a+b) \pmod n$$

Der gælder følgende regler for regning med restklasser:
1) $(a+b) \pmod n = \big(a \pmod n + b \pmod n\big) \pmod n$
2) $(a-b) \pmod n = \big(a \pmod n - b \pmod n\big) \pmod n$
3) $(a \cdot b) \pmod n = \big(a \pmod n \cdot b \pmod n\big) \pmod n$

Bemærkning. Dette er gennemgået og bevist i projekt 0.4. Læg mærke til, at vi på højre side må reducere en ekstra gang modulo *n*, fordi vi evt. kommer uden for tallene i $\mathbb{Z}_n$

Øvelse 12
Vælg et tal *n* og demonstrer regnereglerne med nogle taleksempler.
Alle værktøjsprogrammer har moduloregning indbygget. Skal man udregne 100(mod 6), dvs resten man får ved division af 100 med 6, så skriver man fx

- i TI Nspire: mod(100,6), og får resultatet 4
- i Maple: 100 mod 6, og får resultatet 4

Øvelse 13
a) Bestem 12345 mod 17
b) Bestem 2205^{157} mod 3149

Undersøgelsen af, om to tal er kongruente modulo et tal *n* kan udføres på en lidt anden måde, end ved at opskrive divisionsligningen, nemlig ved at undersøge, om forskellen på de to tal er delelig med *n*. Dette er indholdet i næste sætning. Man kan ofte se denne egenskab anvendt som definition på kongruens.

Sætning 2
1) Hvis $a \pmod n = b \pmod n$, så gælder: $n \mid (a-b)$
2) Hvis $n \mid (a-b)$, så gælder: $a \pmod n = b \pmod n$

3. RSA systemet

RSA systemet bygger på den antagelse, at det er praktisk umuligt at primtalsfaktorisere store tal. Med "store tal" mener vi tal, der har flere hundrede cifre.

RSA-systemet består af:

- en offentlig kendt nøgle *E* (encryption - kryptering), som anvendes af en afsender, der ønsker at kode sin meddelelse, så ingen uvedkommende får adgang. Denne nøgle består af to tal, (nærmere beskrevet nedenfor) som feks kan findes i en telefonbog.
- en hemmelig nøgle *D* (Decryption- dekryptering) som anvendes af modtageren til at afkode meddelelsen. Denne nøgle består af et af de offentlige tal samt et hemmeligt tal (beskrevet nedenfor), og dette skal selvfølgelig opbevares sikkert.

Systemet opfylder kravene til et "smæklås-system":

- det er let at kode (dvs at "låse")
- det er praktisk taget umuligt at finde den oprindelige meddelelse ud fra den kodede;
- der findes en nøgle, således at det for indehaveren af nøglen er en let sag at afkode meddelelsen ("låse op").

Nøglerne findes som følger:

- Vælg to meget store primtal, *p* og *q*, hver med over 100 cifre.
- Udregn $n = p \cdot q$, og $\varphi(n) = (p-1)\cdot(q-1)$. ($\varphi(n)$ kaldes Eulers phi-funktion – den omtales nedenfor)
- Vælg et tal *d*, som er mindre end *n*, men større end både *p* og *q*, og således at *d* er primisk med $\varphi(n)$: (*d*, $\varphi(n)$) = 1
- Beregn *e* som det *inverse* element til *d* modulo $\varphi(n)$, dvs det tal, som opfylder:

$$e \cdot d \equiv 1 \pmod{\varphi(n)}$$

Den offentlige nøgle er så tallene (*n,e*).
Den hemmelige nøgle er tallet *d* (sammen med *n*).
Også tallene *p*, *q*, og $\varphi(n)$ skal holdes hemmelige. Det sikreste er at destruere dem! De har gjort deres pligt og kan gå. Systemet har ikke brug for dem mere; men skulle man miste *d*, ville man selvfølgelig være på spanden. Så kan ingen jordisk sjæl nogensinde trænge ind og afkode meddelelserne - heller ikke afsenderne. Der er ingen bagindgang - informationerne ville for evigt være låst inde.

3.1 Sådan anvendes RSA i praksis

En meddelelse, der ønskes afsendt er oversat til en talrække, feks ved at anvende bogstavernes talværdier: Mellemrum:00, A:01, B:02,..., Å:29; tallene, grammatiske og andre symboler kan naturligvis tilsvarende repræsenteres ved tal-værdier, men lad os nu nøjes med dette.

Nu skal vi først have konstrueret nøglerne!
Lad os vælge nogle små overskuelige primtal, for at vise ideen:

$p = 47$, $q = 67$

Vi udregner:

$n = p \cdot q = 3149$ $\quad\quad$ $\varphi(n) = (p-1)\cdot(q-1) = 3036$

For den hemmelige kode *d* er der mange muligheder, men et stort tal giver ekstra sikkerhed. Vi vælger *d* = 1489. (Check at (*d*, $\varphi(n)$) = (1489,3036) = 1).
Den offentlige nøgle beregnes fx med anvendelse af et værktøsprogram. Vi får *e* = 157.

I Maple kan det se således ud:

$igcdex(1489, 3036, 's', 't'), s, t = 1, 157, -77$

hvilket betyder, at største fælles divisor er 1, og at vi har linearkombinationen:

$157 \cdot 1489 - 77 \cdot 3086 = 1$

Den offentlige nøgle består af tallene (*n,e*) = (3149,157).
Den hemmelige nøgle består af tallet *d* = 1489 (sammen med 3149).
Læg mærke til, at $e \cdot d = 157 \cdot 1489 \equiv 1 \pmod{3086}$

Talrækken opdeles i blokke af en længde, der er mindre end eller lig med n's længde, dvs højst 4 i dette tilfælde. Ingen talblok må være større end tallet 3149, for da vi regner med rester modulo 3149, vil der nemlig så kunne opstå tvetydigheder - vi ville ikke nødvendigvis komme tilbage til det rigtige tal igen.
En sådan talblok *m* krypteres nu til et nyt tal *c* ved at udregne:

$c = m^e \pmod{n}$, dvs $c = m^{157} \pmod{3149}$

Modtageren af de nu uforståelige talblokke dekrypterer ved for hvert sådant *c* at udregne:

$c^d \pmod{n}$, dvs $c^{1489} \pmod{3149}$

og det viser sig nu, at dette tal er lig med m. Altså hjemme igen.

Øvelse 14
Vælg et tilfældigt tal mindre end 3149, fx 2345.
a) Krypter tallet ved at udregne: $c = 2345^{157} \pmod{3149}$
b) Dekrypter det fundne tal, ved at udregne $c^{1489} \pmod{3149}$

Ser vi på nogle realistiske taleksempler, hvor tallene har 100 eller 200 cifre forekommer dette næsten som magi - utroligt men sandt.
Vi skal naturligvis vise, at dette faktisk er tilfældet. Men først demonstreres systemet med nogle små tal, så vi kan se hvordan det virker:

Øvelse 15
Vi krypterer med brug af den omtalte offentlige nøgle og dekrypterer med den hemmelige. Vælg selv en kort tekst, som først oversættes til tal ifølge denne tabel, hvor det første tegn er mellemrum

(-)	a	b	c	d	e	f	g	h	i	j	k	l	m	n
00	01	02	03	04	05	06	07	08	09	10	11	12	13	14

o	p	q	r	s	t	u	v	w	x	y	z	æ	ø	å
15	16	17	18	19	20	21	22	23	24	25	26	27	28	29

Rigtige tekster består naturligvis også af store bogstaver, grammatiske tegn og tal, men vi nøjes med dette.
Oversættelsen til tal kan fx ske med de såkaldte ASCII koder.
a) Oversæt din tekst til talkoder, og opdel i talblokke med 4 cifre i hver.
b) Krypter din tekst.
c) Giv den til din sidemand, som skal prøve at dekryptere den.

Øvelse 16
Vi har fundet to primtal p = 467, q = 479, der er brugt til at lave en RSA-kode. Du ved, at den offentlige nøgle er e = 73443. Beregn den hemmelige nøgle.

Øvelse 17
Nu skal du udfordre din sidemand: I vælger begge to primtal *p* og q, mindre end 1000, som den anden ikke ser. I udregner som i starten af afsnittet for hver jeres system følgende to tal:

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

I vælger begge et tal *d* til jeres hemmelige nøgle, som vi gjorde i starten af afsnittet og I beregner den offentlige nøgle, som I offentliggør sammen med tallet *n*.
Skriv hver en kort tekst, som I dernæst krypterer. Giv kryptoteksten til den anden og se, hvem der først bryder koden og dekrypterer teksten.

4. Beviset for at RSA virker.

I sidste afsnit præsenterede vi RSA-systemet, og vi gennemførte en række øvelser, der demonstrerede, at det tilsyneladende virker. Men det er jo ikke et bevis for, at det vil virke i alle tilfælde.

Vi indfører først det centrale værktøj for RSA-kryptografien, nemlig den såkaldte *Eulers φ-funktion*. (Atter engang Euler: uanset hvilken gren af matematikken man arbejder med, støder man altid ind i Euler).

4.1. Eulers φ−funktion.

Definition.
Mængden af tal mellem 1 og *n*-1, som er *primiske* med n, betegnes Z_n^*. *Antallet* af elementer i Z_n^* betegnes med φ(*n*), hvor φ kaldes for Eulers φ-funktion.

Husk: Et tal er primisk med *n*, hvis de ingen fælles primfaktorer har.

Eksempel 10

$Z_2 = \{0,1\}$	$Z_2^* = \{1\}$	$\varphi(2) = 1$
$Z_3 = \{0,1,2\}$	$Z_3^* = \{1,2\}$	$\varphi(3) = 2$
$Z_4 = \{0,1,2,3\}$	$Z_4^* = \{1,3\}$	$\varphi(4) = 2$
$Z_5 = \{0,1,2,3,4\}$	$Z_5^* = \{1,2,3,4\}$	$\varphi(5) = 4$
$Z_6 = \{0,1,2,3,4,5\}$	$Z_6^* = \{1,5\}$	$\varphi(6) = 2$
$Z_7 = \{0,1,2,3,4,5,6\}$	$Z_7^* = \{1,2,3,4,5,6\}$	$\varphi(7) = 6$

Øvelse 19
Af ovenstående eksempel får vi udfyldt en del af skemaet nedenfor. Udfyld selv resten:

n	2	3	4	5	6	7	8	9	10	11	12
φ(*n*)	1	2	2	4	2	6					

φ(*n*) er stort set lige så svær at få styr på, som primtallene. Den som kan finde en forskrift for φ eller en måde at udregne φ(n*)* på, som ikke bygger på et kendskab til *n*'s primfaktorer, er sikker på at få sit navn udødeliggjort og til evig tid at indtage en hædersplads i matematikhistorien! Men kryptografer vil forbande ham, for det er netop φ(*n*) 's uberegnelige opførsel, den moderne kryptografi er bygget op over.

Kender vi *n*'s primfaktorer kan vi imidlertid også beregne $\varphi(n)$. Vi formulerer to udgaver af denne sætning, henholdsvis hvor *n* har én primfaktor (altså selv er et primtal), og hvor *n* indeholder to primfaktorer.

Sætning.
Hvis *p* er et primtal, er $\varphi(p) = (p-1)$

Bevis.
$Z_p = \{0,1,2,...,p\text{-}1\}$, og da *p* er et primtal, er alle tallene 1,2,...*p*-1 primiske med *p*. Der er således i alt *p*-1 tal i Z_p^*, altså $\varphi(p) = p\text{-}1$.

Sætning.
Hvis *p* og *q* er to forskellige primtal, og *n* = *pq*, så gælder:
$$\varphi(n) = (p-1)\cdot(q-1)$$

Bevis.
$\varphi(n)$ er antallet af tal fra 1 til *n*-1, der er primiske med *n*. Der er i alt *n*-1 tal. Lad os prøve at tælle hvor mange der *ikke* er primiske med *n*. Dette er altså tal, der *har* en fælles divisor med *n* = *pq*.
En sådan fælles divisor må *enten* være *p eller q*. (Husk: en fælles divisor er et produkt af de fælles primfaktorer). Hvor mange tal mellem 1 og *n*-1 indeholder p som primfaktor? Det gør tallene *p,*2*p,*3*p,...,(q*-1*)p*.

Tilsvarende er primfaktoren q indeholdt i tallene: *q,*2*q,*3*q,...,(p*-1*)q*.
Svaret på, hvor mange, der *ikke* er primisk med *n* er altså: *q*-1 + *p*-1.
Antallet der *er* primisk med *n,* altså tallet $\varphi(n)$, er derfor:
$\varphi(n) = (n\text{-}1) - (q\text{-}1) - (p\text{-}1) = n - q - p + 1$.
Indsæt *n* = *pq*: $\varphi(n) = pq - q - p + 1 = (p\text{-}1)(q\text{-}1)$
Det sidste kan kontrolleres ved at gange parentesen ud.

Eksempel 11

a) $10 = 2\cdot 5$ så: $\varphi(10) = 1\cdot 4 = 4$
b) $119 = 7\cdot 17$ så: $\varphi(119) = 6\cdot 16 = 96$

Øvelse 20
Det er vigtigt at lægge mærke til, at sætningen kun gælder for primtal der er *forskellige*. Hvis der er tale om et primtalskvadrat $n = p^2$, så er $\varphi(n) = p\cdot(p-1)$.
Dvs der gælder: $\varphi(4) = 2$, $\varphi(9) = 6$, $\varphi(25) = 20$.
Dette kan vises med samme teknik som sætningen ovenfor. Prøv selv!
Man vil naturligvis ikke vælge et kvadrattal, når man skal lave hemmelige koder, så vi har ingen praktisk interesse i dette.

Øvelse 21
Find tallene: a) $\varphi(77)$ b) $\varphi(221)$

Vi er nu i stand til at vise en anden af talteoriens hovedsætninger:

Eulers sætning.
Hvis $(a,n) = 1$, så gælder:

$$a^{\varphi(n)} \equiv 1 \pmod{n}$$

dvs tallet $a^{\varphi(n)}$ har rest 1 ved division med n.

Før vi viser sætningen ser vi lige på et eksempel:

Eksempel 12

a) 7 og 10 er primiske. Vi ved at $\varphi(10) = 4$. Sætningen påstår så at:

$$7^4 \equiv 1 \bmod 10$$

Ved udregning ser vi: $7^4 = 2401$, som vitterlig har rest 1 ved division med 10.

b) 4 og 13 er primiske. Da 13 er et primtal er $\varphi(13) = 12$. Sætningen påstår, at:

$$4^{12} \equiv 1 \bmod 13$$

Ved udregning får vi: $4^{12} = 16.777.216 = 129055 \cdot 13 + 1$, altså rest 1 ved division med 13.

Bevis for Eulers sætning.

Vi har givet to tal a og n, og vi ved de er primiske: $(a,n) = 1$.

Vi skal vise: $a^{\varphi(n)} \equiv 1 \bmod n$

Der er $\varphi(n)$ tal, som er primiske med n og mindre end n. Lad os opskrive dem:

$$Z_n^* = \left\{r_1, r_2, ..., r_{\varphi(n)}\right\} \qquad (*)$$

Konstruer nu en ny stribe tal ved at gange a på alle disse:

$$\left\{a \cdot r_1, a \cdot r_2, ..., a \cdot r_{\varphi(n)}\right\}$$

Ethvert af disse tal $a \cdot r_i$ må være primisk med n, fordi ingen af de to tal $a \cdot r_i$ og r_i har fælles primfaktorer med n. Hvis vi nu reducerer modulo n, vil tallene derfor ligge i Z_n^*:

$$\left\{a \cdot r_1 \pmod{n}, a \cdot r_2 \pmod{n}, ..., a \cdot r_{\varphi(n)} \pmod{n}\right\} \qquad (**)$$

Disse tal er samtidig alle forskellige. For hvis to var ens, feks:

$$a \cdot r_i \pmod{n} = a \cdot r_j \pmod{n}$$

ville $a \cdot r_i - a \cdot r_j$ være delelig med n, dvs:

$$n \mid (a \cdot r_i - a \cdot r_j) \text{, eller: } n \mid a \cdot (r_i - r_j)$$

Men da $(a,n) = 1$, må der så gælde:

$$n \mid (r_i - r_j)$$

Begge tal r_i og r_j er positive og mindre end n, så derfor må der gælde:

$$n \mid (r_i - r_j) \Rightarrow r_i - r_j = 0 \Rightarrow r_i = r_j$$

Altså har vi set:

$$a \cdot r_i \pmod{n} = a \cdot r_j \pmod{n} \Rightarrow r_i = r_j$$

Nu ved vi derfor, at den nye stribe tal vi lavede (**), er i alt $\varphi(n)$ forskellige tal i Z_n^*.

Men det betyder jo, at vi i (**) lige præcis har alle tal i Z_n^*.

Hvis vi derfor danner *produktet* af tallene i (*) og i (**), så må disse to produkter være ens:

$$r_1 \cdot r_2 \cdot \ldots \cdot r_{\varphi(n)} = \left(a \cdot r_1 (\text{mod } n)\right) \cdot \left(a \cdot r_2 (\text{mod } n)\right) \cdot \ldots \cdot \left(a \cdot r_{\varphi(n)} (\text{mod } n)\right)$$

Dette tal er klart meget større end *n*, og for at finde dets repræsentant i restklassen reducerer vi nu modulo *n*. Vi bruger regnereglerne for moduloregning, og får af denne ligning:

$$\left(r_1 \cdot r_2 \cdot \ldots \cdot r_{\varphi(n)}\right)(\text{mod } n) = \left(\left(a \cdot r_1 (\text{mod } n)\right) \cdot \left(a \cdot r_2 (\text{mod } n)\right) \cdot \ldots \cdot \left(a \cdot r_{\varphi(n)} (\text{mod } n)\right)\right) (\text{mod } n)$$

$$\left(r_1 \cdot r_2 \cdot \ldots \cdot r_{\varphi(n)}\right)(\text{mod } n) = \left(\left(a \cdot r_1\right) \cdot \left(a \cdot r_2\right) \cdot \ldots \cdot \left(a \cdot r_{\varphi(n)}\right)\right) (\text{mod } n)$$

$$\left(r_1 \cdot r_2 \cdot \ldots \cdot r_{\varphi(n)}\right)(\text{mod } n) = \left(a^{\varphi(n)} \cdot r_1 \cdot r_2 \cdot \ldots \cdot r_{\varphi(n)}\right) (\text{mod } n)$$

Dette kan også udtrykkes som at *n* går op i differensen af de to tal:

$$n \mid \left(a^{\varphi(n)} \cdot r_1 \cdot r_2 \cdot \ldots \cdot r_{\varphi(n)} - r_1 \cdot r_2 \cdot \ldots \cdot r_{\varphi(n)}\right), \text{ eller:}$$

$$n \mid \left((a^{\varphi(n)} - 1) \cdot r_1 \cdot r_2 \cdot \ldots \cdot r_{\varphi(n)}\right)$$

n er primisk med alle *r*'erne, og derfor også med deres produkt. Men så ved vi jo, at *n* må gå op i den anden faktor:

$$n \mid (a^{\varphi(n)} - 1), \text{ eller som sætningen siger:}$$

$$a^{\varphi(n)} \equiv 1 \ (\text{mod } n)$$

Øvelse 21

Vis, at Fermats lille sætning er et specialtilfælde af Eulers sætning:

Hvis p er et primtal, og (a,p) = 1, så gælder $a^{p-1} \equiv 1 \ (\text{mod } p)$

Eulers sætning vil hjælpe os med til at kunne *afkode* en meddelelse, der er kodet til noget - for andre - helt uforståeligt.

Af Eulers sætning får vi en lille teknisk udseende sætning, som viser, hvordan man reducerer potenser modulo n, som vi får brug for til det endelige bevis for, at RSA-systemet virker:

Sætning.
For tallene *a* og *n*, hvor (*a,n*) = 1, gælder for alle tal *r*:

$$a^r (\text{mod } n) = a^{r(\text{mod } \varphi(n))} (\text{mod } n)$$

Bevis.

Opskriv divisionsligningen for *r* og φ(*n*):

$$r = k \cdot \varphi(n) + r(\text{mod } \varphi(n))$$

Vi indsætter dette udtryk for *r* i venstre side af den ligning, som vi skal vise gælder, og regner løs idet vi udnytter regnereglerne for modulo-regning:

$$\begin{aligned} a^r \pmod n &= a^{k\cdot\varphi(n)+r(\bmod\ \varphi(n))} \pmod n \\ &= a^{k\cdot\varphi(n)} \cdot a^{r(\bmod\ \varphi(n))} \pmod n \\ &= \left(a^{k\cdot\varphi(n)} \pmod n \cdot a^{r(\bmod\ \varphi(n))} \pmod n\right) \pmod n \\ &= \left((a^{\varphi(n)})^k \pmod n \cdot a^{r(\bmod\ \varphi(n))} \pmod n\right) \pmod n \\ &= \left((a^{\varphi(n)} \pmod n)^k \cdot a^{r(\bmod\ \varphi(n))} \pmod n\right) \pmod n \\ &= \left((1)^k \cdot a^{r(\bmod\ \varphi(n))} \pmod n\right) \pmod n \end{aligned}$$

Bemærk, at i den sidste omskrivning udnytter vi Eulers sætning. Da $(1)^k = 1$ får vi altså:

$$a^r \pmod n = a^{r(\bmod\ \varphi(n))} \pmod n$$

Vi har regnet os frem fra venstre side til højre side af ligningen. Så sætningen gælder.

Eksempel.
Find resten ved division af 3^{523} med 7. Dvs find ud af hvad 3^{523} (mod 7) er?
Tallet 3^{523} er stort, men ikke større end at værktøjsprogrammer kan udregne det:

3^{523} =
342307345274048444418587191189960272522120345527
0744842679540287347185276727273132942177916\
6830574204213991438754804731197942867677382\
4787953155766853423090459737745394076361756\
5406227498787581128475121030114571406089675\
25372214963526316006648827

Programmerne svarer også uden tøven, at 3^{523} (mod 7) = 3.
Programmerne har givetvis indbygget algoritmer, der anvender sætningen ovenfor. Vi kan faktisk let udregne det i hånden:
3 og 7 er primiske, $\varphi(7) = 6$, og 523 (mod 6) = 1.
Sætningen giver nu:

$$3^{523} \pmod 7 = 3^{523\ (\bmod\ \varphi(7))} \pmod 7 = 3^{523(\bmod\ 6)} \pmod 7 = 3^1 \pmod 7 = 3$$

Øvelse 22
a) Find resten ved division af 7^{3527} med 9
b) Find resten ved division af 12^{252971} med 11

Betegnelser .
Når vi regner modulo n, altså kun interesserer os for *resten* ved division med *n*, så siger vi, at et tal *b* er *inverst* eller *omvendt* til *a* (med hensyn til *n*), hvis der gælder:

$$a \cdot b \equiv 1 \pmod n$$

Bemærkning,
Selv om det ikke udtrykkeligt nævnes hver gang, er det vigtigt at huske, at den *inverse* til et a afhænger af *n*; vi ser jo kun på *rester*, og resten vil ændre sig, hvis vi vælger et andet tal end *n* som divisor.

Hvis (*a,n*) = 1, og hvis vi kender $\varphi(n)$ så har vi en *formel* for den inverse (modulo n). Vi ved nemlig fra Eulers sætning, at:

$$a^{\varphi(n)} \equiv 1 \pmod{n}$$

eller: $$a \cdot a^{\varphi(n)-1} \equiv 1 \pmod{n}$$

Så: $a^{\varphi(n)-1}$ er inverst til *a*.

Da $\varphi(n)$ kan være vanskelig at finde, så er det ofte mere fremkommeligt at anvende *Euklids algoritme* til at finde *s* og *t*, så:

$$s \cdot a + t \cdot n = 1$$

eller: $$s \cdot a = -t \cdot n + 1$$

Her står jo at læse, at $s \cdot a$ har rest 1 ved division med n:

$$s \cdot a \equiv 1 \bmod n$$

eller med andre ord: *s* er inverst til *a*.
Vi sammenfatter dette i:

Sætning.
Hvis *a* og *n* er primiske, så har a (mod n) et inverst element s.
s kan findes ved brug af Euklids algoritme.

Øvelse 23
Vis, at der i intervallet fra 1 til n-1 kun findes ét inverst element til a, hvor a er primisk med n.
Vi kan således tillade os at tale om *det* inverse element til a.

Eksempel.
a) 5 er primisk med 119. Vi ønsker at bestemme det inverse element til 5. Vi bruger Euklids algoritme og anvender et værktøjsprogram til at finde:

$$24 \cdot 5 + (-1) \cdot 119 = 1$$

Heraf aflæses, at 24 er det inverse element til 5 modulo 119

b) 41 er primisk med 293. Vi ønsker at bestemme det inverse element til 41. Vi bruger igen Euklids algoritme og anvender et værktøjsprogram til at finde:

$$(-50) \cdot 41 + 7 \cdot 293 = 1$$

Heraf aflæses, at -50 er et inverst element til 41. Vi ønsker at finde et indenfor Z_{293}^{*}, og lægger derfor 293 til -50. Så bliver vi i samme restklasse, men lander indenfor de principale rester. Det inverse element til 41 er derfor 243.

Øvelse 24
a) 280 er primisk med 8613. Find den inverse til 280 modulo 8613.
b) 19 er primisk med 4879. Find den inverse til 19 modulo 4879.

4.2 RSA virker

Antag nu, vi har den situation, vi beskrev i starten af afsnit 3:
Vi har

- valgt to meget store primtal, *p* og *q*, hver med over 100 cifre.
- udregnet $n = p \cdot q$, og $\varphi(n) = (p-1) \cdot (q-1)$.
- valgt et tal *d*, som er mindre end *n*, men større end både *p* og *q*, og således at *d* er primisk med $\varphi(n)$: (*d*, $\varphi(n)$) = 1
- beregnet *e* som det *inverse* element til *d* modulo $\varphi(n)$, dvs det tal, som opfylder:

$$e \cdot d \equiv 1 \pmod{\varphi(n)}$$

Vi påstod da, at:
Som offentlig nøgle kan vi anvende tallene (*n,e*).
Som hemmelig nøgle kan vi anvende tallet *d* (sammen med *n*).

Påstanden er nu, at hvis en talblok *m* krypteres til et nyt tal *c* ved at udregne:

$$c = m^e \pmod{n},$$

og hvis modtageren af de nu uforståelige talblokke dekrypterer ved for hvert sådant *c* at udregne:

$$c^d \pmod{n},$$

så vil det vise sig, at dette tal er lig med m.
Dette vil vi nu bevise er sandt! Dvs vi vil bevise, at

$$m^{ed} \left(\text{mod } n\right) = m$$

Vi ved, at $e \cdot d \equiv 1 \pmod{\varphi(n)}$, hvilket betyder, at der findes et helt tal k, så:

$$e \cdot d = k \cdot \varphi(n) + 1 \qquad (*)$$

Vi ved også, at $\varphi(n) = (p-1) \cdot (q-1)$, som vi kan indsætte:

$$e \cdot d = k \cdot \varphi(n) + 1 = k \cdot (p-1) \cdot (q-1) + 1 \qquad (**)$$

Vi skelner nu mellem to tilfælde:
1) *m* og *n* er primiske 2) *m* og *n* er ikke primiske.

1) ***m og n er primiske: (m,n) = 1***. (Bemærk at dette er langt det mest almindelige - hvorfor?).
Vi omskriver udtrykket $m^{ed} \pmod{n}$:

$m^{ed} \left(\text{mod } n\right)$ indsæt (*):

$m^{k \cdot \varphi(n)+1} \left(\text{mod } n\right)$ anvend potensregler:

$\left(m^{\varphi(n)}\right)^k \cdot m^1 \left(\text{mod } n\right)$ anvend Eulers sætning:

$(1)^k \cdot m^1 \left(\text{mod } n\right)$

$m \left(\text{mod } n\right)$

Altså hjemme igen.

2) ***m og n er ikke primiske.***
En fælles divisor må være enten p eller q, feks p:

$p \mid m$ Så må der også gælde:

$p \mid m^{ed}$

Derfor findes et helt tal h, så:

$$m^{ed} - m = h \cdot p \qquad (***)$$

q og m må være primiske, for ellers gik q op i m, og dermed ville *n = pq* gå op i m, men m er mindre end n, så det går ikke.

Når q og m er primiske får vi af Fermats lille sætning:

$$m^{q-1} \equiv 1 \bmod q$$

Vi vil nu bruge (**) og omskriver derfor:

$$\begin{aligned}
&m^{q-1} \equiv 1 \bmod q && \Rightarrow \\
&\left(m^{q-1}\right)^{(p-1)} \equiv (1)^{(p-1)} \bmod q && \Rightarrow \\
&m^{(q-1)(p-1)} \equiv 1 \bmod q && \Rightarrow \\
&m^{\varphi(n)} \equiv 1 \bmod q && \Rightarrow \\
&\left(m^{\varphi(n)}\right)^{k} \equiv 1^{k} \bmod q && \Rightarrow \\
&m^{k\cdot\varphi(n)} \equiv 1 \bmod q && \Rightarrow \\
&m^{k\cdot\varphi(n)} \cdot m \equiv 1 \cdot m \bmod q && \Rightarrow \\
&m^{k\cdot\varphi(n)+1} \equiv m \bmod q && \Rightarrow \\
&m^{ed} \equiv m \bmod q &&
\end{aligned}$$

Den sidste kongruens fortæller, at der findes et helt tal *l*, så:

$$m^{ed} - m = l \cdot q \qquad (****)$$

Samlet har vi derfor:

$$m^{ed} - m = h \cdot p \qquad (***)$$

$$m^{ed} - m = l \cdot q \qquad (****)$$

Heraf får vi: $h \cdot p = l \cdot q$

Da p og q er forskellige primtal, må der så gælde:

$q \mid h$, dvs.:

der findes et helt tal *r* så:

$$h = r \cdot q$$

Indsæt dette i (****):

$$m^{ed} - m = h \cdot p = (r \cdot q) \cdot p = r \cdot q \cdot p = r \cdot n$$

Her strår, at *n* går op i $m^{ed} - m$, eller at:

$$m^{ed} \pmod n = m$$

Altså også i dette tilfælde ender vi med det tal *m* vi startede med.

Konklusion: I begge tilfælde virker RSA: En given talblok der krypteres med den offentlige nøgle kan dekrypteres med den hemmelige, eller omvendt.

5. Afslutning

En hovedårsag til at RSA-systemet endnu ikke bruges i stor udstrækning er, at det trods stærke regnemaskiner og gode algoritmer tager for lang tid at udregne de gigantiske tal.

Et stort teoretisk problem i udarbejdelsen af en RSA-nøgle er om vi kan vide med sikkerhed, om et givet tal er et primtal eller ej. Det er svært, når det er tal med 100 cifre. Vi har tidligere set, at der er så mange primtal, at det ikke skulle være så svært at finde nogen. Men hvordan kan vi være sikre? Det kan vi heller ikke. En fuldstændig undersøgelse er håbløst tidkrævende - det er jo netop hele fidusen. Derfor er der udviklet en lang række såkaldte *primtals-test*. Kan et tal stå for en række af disse test, accepteres det som primtal; men sikker er vi aldrig.

Et særligt sikkerhedsmæssigt problem er at vælge sådanne primtal, at koden bliver sværest muligt at bryde. Og her er ikke alle primtal lige gode. Dels skal p og q være store, ca 100 cifre hver; men de må heller ikke ligge for tæt ved hinanden, for et umiddelbart angreb på systemet ville være at udregne √n og så prøve sig frem her ud fra.
Men ud over dette kan man vise, at systemet bliver ekstra sikkert, hvis de valgte primtal er såkaldt "stærke" primtal, hvilket dækker over følgende: Et primtal p kaldes *stærkt* hvis:

- p-1 og p+1 begge har en stor primfaktor (der vil være svær at finde)
- for mindst én af disse primfaktorer s gælder også, at s-1 har en stor primfaktor.

Hvis både p og q er primtal der opfylder disse krav, så er systemet rimeligt sikkert.

RSA-systemet har et fortrin frem for alle andre systemer, nemlig at man ved hjælp af dette kan *underskrive* dokumenter, på en måde så underskriften ikke kan forfalskes. Dette er indholdet i begrebet *digital signatur*. Det har hidtil været opfattet som en grundlæggende svaghed ved elektronisk kommunikation: Hvordan kan vi være sikre på, at en meddelelse, vi modtager, virkelig er fra den, som det fremgår af meddelelsen. Kan der ikke være tale om at en eller anden svindler sender os noget, og så udgiver sig for at være en anden?

Hvis to institutioner eller personer A og B vil kommunikere, så de er helt sikre på, at ingen kan læse deres meddelelse, og at de selv ikke er i tvivl om, hvorfra meddelelsen kom, så skal de blot begge udnytte deres personlige hemmelige kode, samt de tilsvarende offentlige koder:

Når A vil sende en besked til B, så kodes beskeden først med B's offentlige nøgle; herefter kan ingen andre end B læse den. Derefter kodes den endnu engang med A's egen hemmelige nøgle. Der er ingen i verden undtagen A, som kan kode på denne måde, så nu har beskeden fået A's fingeraftryk. B modtager den, og afkoder, først ved brug af A's offentlige kode - så er vi inde ved det der er kodet med B's offentlige nøgle, og B anvender nu sin egen hemmelige nøgle til at lukke op for den sidste indpakning. Hvis der nu står noget læseligt, så kan det kun være sendt fra A.

Øvelse 25
Overvej hvordan man kan bruge denne egenskab med digital signatur, til at slå plat og krone, eller spille poker via elektronisk kommunikation, således at vi er sikre på, at modparten ikke snyder.

Nogle svar:
Øvelse 1 e): $2^3 \cdot 3^2 \cdot 19$
Øvelse 1 f) $3^2 \cdot 13 \cdot 17^3$
Øvelse 16: Den hemmelige nøgle: d = 70.167;
Øvelse 22a): $\varphi(9) = 6$, $3527 \pmod 6 = 5$, og $7^5 \pmod 9 = 7^2 \cdot 7^2 \cdot 7 \pmod 9 = 4 \cdot 4 \cdot 7 \pmod 9 = 4$
Øvelse 22b): $\varphi(11) = 10$, $12^{252971} \pmod{11} = 12^{252971 \pmod{10}} \pmod{11} = 12^1 \pmod{11} = 12 \pmod{11} = 1$