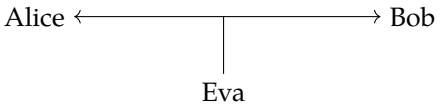
Kapittel 28

Kryptologi

I dette kapitlet lærer vi hvordan tallteori kan anvendes til å sikre kommunikasjon. Vi lærer om symmetrisk kryptografi, nøkkelutveksling, og offentlig-nøkkel kryptografi, inkludert Diffie–Hellman og RSA.

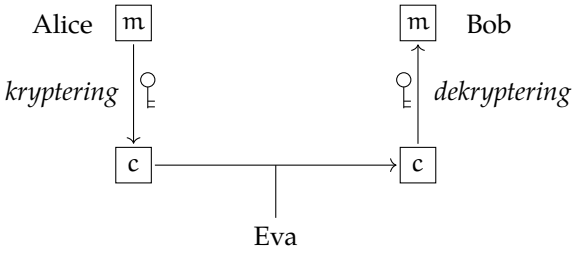
Kryptering av meldinger

To personer, Alice og Bob, vil kommunisere med hverandre (for eksempel via brev eller epost). Men en tredje person, Eva, kan observere alle meldingene som sendes.



Alice og Bob vil ikke at Eva skal vite hva de skriver til hverandre. Derfor vil de gjøre om hver melding som skal sendes til en hemmelig form, slik at det bare er de selv som kan få tilbake den opprinnelige meldingen. Kryptografi handler om teknikker for å få til dette.

Når Alice vil sende en melding m til Bob, må hun først *kryptere* den. Da bruker hun en *krypteringsfunksjon* og en hemmelig *nøkkel*. Nøkkelen består av et eller flere tall. Ved å bruke krypteringsfunksjonen på meldingen og nøkkelen, får hun en kryptert melding c. Denne sender hun til Bob. Han mottar c, og bruker en *dekrypteringsfunksjon* sammen med den hemmelige nøkkelen for å gjøre om c til den opprinnelige meldingen m.



Vi antar at Eva kjenner til funksjonene som Alice og Bob bruker til kryptering og dekryptering, men ikke nøkkelen. Denne antagelsen er kjent som Kerckhoffs' prinsipp, og sier at sikkerheten til et system skal kun avhenge av at nøkkelmaterialet hemmeligholdes, og *ikke* at selve krypteringsmetoden er hemmelig. Så selv om Eva observerer den krypterte meldingen c, og kjenner metoden som ble brukt til å produsere den, kan hun ikke få tak i den opprinnelige meldingen m, siden hun ikke har nøkkelen som trengs for å dekryptere.

Et *kryptosystem* består av en krypteringsfunksjon, en tilhørende dekrypteringsfunksjon og en (vanligvis veldig stor) mengde med mulige nøkler.

Eksempel Her er et veldig enkelt kryptosystem: Krypteringsfunksjonen erstatter hver bokstav med den som er k plasser senere i alfabetet, og dekrypteringsfunksjonen erstatter hver bokstav med den som er k plasser tidligere i alfabetet. Nøkkelen er tallet k. Dette er det såkalte Cæsarchifferet.

Anta at Alice og Bob har blitt enige om nøkkelen $k = 2$, og at Alice vil sende meldingen «`HEI`» til Bob. Hun krypterer meldingen og får «`JGK`», som hun sender. Bob mottar den krypterte meldingen, og dekrypterer den til «`HEI`».

Eva ser bare den krypterte meldingen «`JGK`», og vet ikke hvilken nøkkel som skal brukes for å dekryptere, så hun får ikke lest den virkelige meldingen.

(Merk: Dette systemet er ikke sikkert. Siden det er veldig få mulige nøkler, kan Eva bare prøve alle sammen, og se hvilken av dem som gir et meningsfylt resultat. I et virkelig kryptosystem må det finnes enormt mange mulige nøkler, slik at det er praktisk umulig for Eva å prøve alle.) ♦

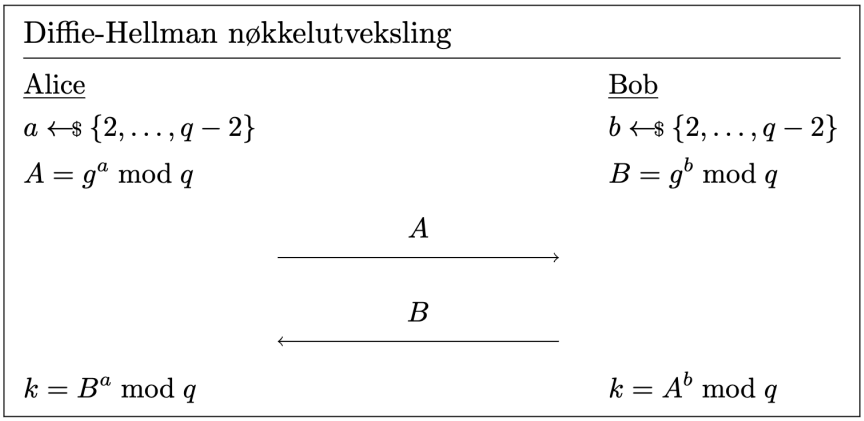
I den generelle situasjonen vi beskrev over, og i eksemplet, var det samme nøkkel som ble brukt både til kryptering og dekryptering. Dette kalles *symmetrisk kryptografi*. Når Alice og Bob skal kommunisere med et symmetrisk kryptosystem, må de først ha blitt enige om en felles hemmelig nøkkel. Det er ikke alltid det er praktisk mulig å få til dette.

Dette er problemet som *asymmetrisk kryptografi* løser, som vi skal fokusere på i resten av dette kapitlet.

Nøkkelutveksling

Målet med en nøkkelutveksling er at to personer skal bli enige om en hemmelig nøkkel som skal kunne brukes i et symmetrisk kryptosystem. Vi så ovenfor at vi kan kryptere en melding dersom begge parter har en delt nøkkel. Cæsarchifferet er selvfølgelig ikke sikkert, men vi har systemer som er bygget på samme konseptet som er sikre å benytte. For å finne mer informasjon om dette kan vi anbefale å søke opp *Advanced Encryption Standard* på Wikipedia.

Hvordan kan Alice og Bob bli enige om en hemmelig nøkkel dersom Eva kan lese alle meldingene de sender? Dette er et veldig interessant problem å



studere. Tanken er at ved hjelp av spesielle matematiske problemer som er *enveisfunksjoner*, så kan vi oppnå dette. En enveisfunksjon er en funksjon som er enkel å beregne, men som er vanskelig å invertere. Vi har tidligere sett at det er effektivt å beregne modulære potenser, men at det er vanskelig å beregne diskrete logaritmer og å faktorisere produkter av store primtall. Vi skal benytte førstnevnte til å lage en protokoll for nøkkelutveksling.

Diffie-Hellman

Protokollen er ganske enkel. Alice og Bob blir enige om noen offentlige parametre som alle får tilgang til. Disse parametrene er et primtall q og en base g. Så er trikset at både Alice og Bob trekker hver sitt tilfeldige tall a og b mellom 2 og $q-2$ (fordi dersom vi velger 0 så er $g^0 = 1$, velger 1 så er $g^1 = g$ og velger $q-1$ så er $g^{q-1} = 1$ modulo q). Målet er at Alice og Bob ender opp med en delt nøkkel $k = g^{a \cdot b} \bmod q$, men at denne nøkkelen er umulig for Eva å få tak i.

Vi observerer at dersom Alice velger en tilfeldig a, så er det vanskelig for Eva å finne a dersom Alice publiserer $A = g^a \bmod q$. Tilsvarende er det vanskelig for Eva å finne b dersom Bob publiserer $B = g^b \bmod q$ dersom b er trukket tilfeldig. Men dersom Alice og Bob sender A og B til hverandre, kan de enkelt beregne den delte nøkkelen uten å lære den andre sin hemmelige potens på følgende vis: $k = g^{a \cdot b} = A^b = B^a \bmod q$. Men Eva må vite a eller b for å kunne beregne k, fordi det er ikke mulig å beregne k fra kun A og B alene.

Det følger direkte fra protokollbeskrivelsen at Alice og Bob ender opp med samme nøkkel på grunn av eksponentregler. Vi ser også at protokollen er sikker mot Eva fordi det er vanskelig å beregne diskrete logaritmer. Når Alice og Bob har blitt enige om k så kan de bruke denne nøkkelen til å kryptere meldinger ved å bruke symmetrisk kryptografi som beskrevet tidligere. Dette er faktisk dette vi gjør på Internett i dag når vi kommuniserer sikkert med tjenester som TikTok

og nettbank og lånekassen. Først gjennomfører man en nøkkelutveksling med tjenesten man ønsker å bruke, og så bruker man denne nøkkelen til å kryptere bilder, meldinger, transaksjoner osv. på et sikkert vis.

Eksempel La oss si at Alice og Bob er enige om å bruke primtallet $q = 11$ og basen $g = 2$. Da har vi sett tidligere at vi kan beregne de følgende potensene modulo 11:

$$2^2 \equiv 4, \quad 2^3 \equiv 8, \quad 2^4 \equiv 5, \quad 2^5 \equiv 10,$$
$$2^6 \equiv 9, \quad 2^7 \equiv 7, \quad 2^8 \equiv 3, \quad 2^9 \equiv 6.$$

Dersom Alice trekker sitt hemmelige tall $a = 4$ og Bob trekker sitt hemmelige tall $b = 7$, får vi $A = 2^4 \bmod 11 = 5$ og $B = 2^7 \bmod 11 = 7$. Når de sender tallene til hverandre, kan de beregne den delte nøkkelen som $k = 5^7 \bmod 11 = 7^4 \bmod 11 = 3$. Vi observerer at $4 \cdot 7 \bmod 10 = 8$, og dette stemmer med tabellen vår for 2^8. (Her reduserte vi potensen modulo $q - 1 = 10$, siden vi har at $2^{q-1} \equiv 1 = 2^0 \pmod q$.)

Dette eksempelet er selvfølgelig ikke sikkert i praksis, fordi tallene er for små. Dersom vi benytter et primtall q med mer enn 900 sifre, er det i praksis umulig for Eva å beregne den diskrete logaritmen til A eller B for å finne nøkkelen. ♦

Offentlig nøkkel-kryptografi

En annen form for kryptografi, som kalles *offentlig nøkkel-kryptografi*, bruker ulike nøkler til kryptering og dekryptering, og forutsetter ikke at partene på forhånd velger en felles nøkkel.

I offentlig nøkkel-kryptografi har man istedenfor en enkelt nøkkel et *nøkkelpar* som består av:

- En *offentlig nøkkel* som brukes til kryptering.
- En *privat nøkkel* som brukes til dekryptering.

Hver person har sitt nøkkelpar. Den private nøkkelen må holdes hemmelig og skal ikke deles med noen andre (ikke engang de man vil kommunisere med), mens den offentlige kan man gjøre kjent for absolutt alle.

For eksempel kan Alice ha et nøkkelpar (A_O, A_P), der A_O er den offentlige nøkkelen hennes, og A_P er den private. Hun kan lage nøkkelparet helt på egen hånd, uten å dele noe hemmelig informasjon med noen andre. Hun beholder den private nøkkelen selv, og sender den offentlige nøkkelen til alle som vil ha den. Tilsvarende har Bob et nøkkelpar (B_O, B_P).

Når Alice vil sende en melding m til Bob, må hun først få Bobs offentlige nøkkel B_O. Den kan Bob sende til Alice. Da kan Eva (som avlytter all kommunikasjonen) også få den med seg, men det gjør ikke noe, fordi den offentlige nøkkelen ikke trenger å holdes hemmelig. Alice krypterer meldingen m ved hjelp av nøkkelen B_O og får en kryptert melding c. Den sender hun til Bob, som dekrypterer den med sin private nøkkel B_P, og dermed får tilbake meldingen m.

Eva får tak i både den offentlige nøkkelen B_O og den krypterte meldingen c, men hun kan likevel ikke finne ut hva den opprinnelige meldingen m er, fordi hun ikke har Bobs private nøkkel.

I et offentlig nøkkel-system kan altså alle kryptere meldinger med Bobs offentlige nøkkel, men det er bare Bob som klarer å dekryptere dem.

På samme måte: Når Bob skal sende en melding til Alice, må han bruke den offentlige nøkkelen hennes, A_O. Dermed er Alice den eneste som klarer å dekryptere meldingen han sender til henne.

Kryptosystemet RSA

RSA er et offentlig nøkkel-kryptosystem, og det er et av de eldste systemene av denne typen. Det ble laget i 1977 av Ron **R**ivest, Adi **S**hamir og Leonard **A**dleman, og har navn etter forbokstavene i opphavsmennenes etternavn.

Ideen i RSA er slik: Vi begynner med å velge to forskjellige primtall p og q, og lar $n = pq$ være produktet av dem. Så velger vi et tall e. Vi representerer hver melding ved et tall m som er slik at $0 \leqslant m < n$. Vi vil at krypteringen av m skal være tallet

$$c = m^e \bmod n.$$

Hva må vi gjøre for å dekryptere en slik melding? Altså: Hvordan kan vi få tilbake m fra tallet c? Hvis vi klarer å finne et tall d slik at $m^{ed} \equiv m \pmod{n}$, så får vi $c^d \equiv m^{ed} \equiv m \pmod{n}$. Da kan vi altså bare regne ut $c^d \bmod n$ for å få tilbake m.

Men kan vi finne et slikt tall d? La oss prøve med å la d være en invers til e modulo $\varphi(n)$. Da er

$$ed = l \cdot \varphi(n) + 1$$

for et tall l. Hvis vi antar at m og n er relativt primiske, kan vi bruke Eulers teorem og få

$$m^{ed} = m^{l\cdot\varphi(n)+1} = (m^{\varphi(n)})^l \cdot m \equiv m \pmod{n},$$

som betyr at tallet d har egenskapen vi ønsket. (Her måtte vi anta at m og n er relativt primiske. Senere skal vi gjøre dette litt grundigere, og vise at det fungerer for alle m.)

Den offentlige nøkkelen består av tallene n og e, og den private nøkkelen består av tallene p, q og d. For å kryptere en melding m bruker man funksjonen

$$f(m) = m^e \bmod n,$$

og for å dekryptere en kryptert melding c bruker man funksjonen

$$g(c) = c^d \bmod n.$$

Tallet e kalles *krypteringseksponenten*, og tallet d kalles *dekrypteringseksponenten*.

Eksempel Vi lager et RSA-nøkkelpar. Vi begynner med å velge to forskjellige primtall. La oss velge $p = 7$ og $q = 13$. Da får vi $n = pq = 7 \cdot 13 = 91$, og vi har $\varphi(n) = (7-1)(13-1) = 72$.

Videre velger vi $e = 5$, og vi kan regne ut at 29 er en invers til 5 modulo 72, så vi setter $d = 29$.

Dermed har vi laget et nøkkelpar bestående av den offentlige nøkkelen $(n, e) = (91, 5)$ og den private nøkkelen $(p, q, d) = (7, 13, 29)$.

La oss nå se på hvordan en melding krypteres og dekrypteres. Vi ser på meldingen $m = 24$. Vi bruker krypteringsfunksjonen f og får:

$$c = f(m) = m^e \bmod n = 24^5 \bmod 91 = 33.$$

Meldingen $m = 24$ blir altså kryptert til $c = 33$.

Hvis vi vil dekryptere den krypterte meldingen $c = 33$, bruker vi dekrypteringsfunksjonen g og får:

$$g(c) = c^d \bmod n = 33^{29} \bmod 91 = 24.$$

Dekrypteringsfunksjonen gir oss altså tilbake meldingen vi startet med.

(Her er noen detaljer i utregningene utelatt. For å beregne $24^5 \bmod 91$ og $33^{29} \bmod 91$ bruker vi teknikken for modulær potensregning som vi lærte tidligere.) ♦

Sikkerheten i RSA baserer seg på at kun den som har laget nøkkelparet kjenner faktoriseringen av n. Man trenger denne faktoriseringen for å klare å regne ut $\varphi(n)$, og man trenger tallet $\varphi(n)$ for å klare å regne ut hva d skal være. Selv om Eva vet hva n er, må vi altså anta at hun ikke er i stand til å faktorisere n og finne tallene p og q.

Nøkkelparet vi lagde i eksemplet over gir selvsagt ingen sikkerhet, siden det er veldig lett å faktorisere tallet $n = 91$. For at RSA skal være sikkert, må vi ha veldig store tall, mye større enn vi klarer å regne med for hånd. Når RSA brukes i praksis, må man derfor bruke datamaskiner til å gjøre beregningene. Hvis vi

skulle laget et mer realistisk eksempel, kunne p, q og n for eksempel sett slik ut:

$$\begin{aligned}
p &= 1049126657281564636718854286135157976973423708820062637813183044378229020873974817454036253792138558094260796518104279455518266961975753112040121467686512 7\\
q &= 1040306003428399056319817687630536316298937005388132559622312675643545364865901821969876269934571871583140675458411193290992681903674681987982536062268 1003\\
n &= 1091412759926780222750525415285368794050398390529299354868656651893264663954610886425749314917610916703850437918072262938779805261345059522828134794965951343201240613084211896958397149394630575944973381868473029469167418415138484684405961894908282103557252606928511935279290440183694516462991516294181760823 81
\end{aligned}$$

Med så store tall blir det håpløst å regne for hånd, men disse tallene er likevel små nok til at en datamaskin uten problemer kan gjøre alle beregningene som trengs i RSA. Samtidig er de store nok til at det er praktisk umulig å faktorisere n.

Dette er fordi vi, som vi har sett i de foregående kapitlene, har rimelig *effektive* algoritmer for operasjoner som modulær eksponensiering, mens vi derimot ikke kjenner noen tilsvarende effektive algoritmer for å faktorisere store tall ved hjelp av vanlige datamaskiner.

…Men vi kan ikke bevise at en slik algoritme ikke eksisterer! Det er fullt mulig at den bare ikke har blitt oppdaget enda, og dersom noen publiserte en slik algoritme i morgen, ville all RSA-sikret kommunikasjon vært brutt over natten.

En variant av Eulers teorem

For at RSA skal fungere som ønsket, må vi ha

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

for alle mulige meldinger m. Her er e krypteringsnøkkelen og d dekrypteringsnøkkelen, og disse to er hverandres inverser modulo $\varphi(n)$. Vi så tidligere at dersom $m \perp n$, så gir Eulers teorem at denne kongruensen holder. Men vi vil at den skal holde også dersom m ikke er relativt primisk til n. Vi skal nå løse dette problemet ved å vise et resultat som ligner på Eulers teorem, men er tilpasset akkurat det vi trenger i RSA.

Først: Vi husker at Fermats lille teorem sier at hvis p er et primtall og a er et tall slik at $p \nmid a$, har vi $a^{p-1} \equiv 1 \pmod{p}$. Videre har vi for *alle* heltall a at $a^p \equiv a \pmod{p}$.

Senere lærte vi om Eulers teorem, som sier at hvis a og n er relativt primiske, så er $a^{\varphi(n)} \equiv 1 \pmod{n}$. Kan vi på samme måte som for Fermat over, bruke dette til å vise at vi har $a^{\varphi(n)+1} \equiv a \pmod{n}$ for alle heltall a?

Det kan vi dessverre ikke; her er et moteksempel:

Eksempel La $n = 4$ og $a = 2$. Da har vi $\varphi(4) = 2$, og $2^{\varphi(4)+1} = 2^3 = 8 \equiv 0 \not\equiv 2 \pmod{4}$. ♦

Påstanden $a^{\varphi(n)+1} \equiv a \pmod{n}$ holder altså ikke for alle a og n. Men hvis vi begrenser oss til noen verdier av n, så holder den for alle a.

Definisjon 28.1 Kvadratfrie tall

Et naturlig tall n er *kvadratfritt* hvis det ikke finnes noe primtall p slik at $p^2 | n$.

Altså: Tallet n er kvadratfritt hvis og bare hvis det finnes forskjellige primtall $p_1, p_2, \ldots p_r$ slik at $n = p_1 \cdot p_2 \cdots p_r$. For eksempel er $42 = 2 \cdot 3 \cdot 7$ et kvadratfritt tall, mens $44 = 2^2 \cdot 11$ ikke er kvadratfritt.

Her er det vi vil vise: La n være et kvadratfritt naturlig tall. Da er $a^{\varphi(n)+1} \equiv a \pmod{n}$ for alle heltall a. La $n = p_1 \cdot p_2 \cdots p_r$ være primtallsfaktoriseringen av n. Vi finner $\varphi(n)$ på følgende måte:

$$\varphi(n) = \prod_{i=1}^{r} (p_i - 1).$$

For hver i har vi at $(p_i - 1) | \varphi(n)$, så vi kan skrive $\varphi(n) = (p_i - 1) \cdot s_i$ for et naturlig tall s_i. Hvis $p_i \nmid a$, får vi ved å bruke Fermats lille teorem at

$$a^{\varphi(n)+1} = a^{(p_i-1)\cdot s_i + 1} = (a^{p_i-1})^{s_i} \cdot a \equiv a \pmod{p_i}.$$

Hvis $p_i | a$, så er $a \equiv 0 \pmod{p_i}$, og vi får

$$a^{\varphi(n)+1} \equiv 0 \equiv a \pmod{p_i}.$$

I begge tilfeller har vi altså $a^{\varphi(n)+1} \equiv a \pmod{p_i}$. Siden dette holder for hver i, har vi nå følgende kongruenser:

$$\begin{aligned} a^{\varphi(n)+1} &\equiv a \pmod{p_1} \\ a^{\varphi(n)+1} &\equiv a \pmod{p_2} \\ &\vdots \\ a^{\varphi(n)+1} &\equiv a \pmod{p_r} \end{aligned}$$

Siden tallene $p_1, p_2, \ldots, p_r$ er parvis relativt primiske, og $n = p_1 \cdot p_2 \cdots p_r$, kan vi bruke det kinesiske restteoremet og få:

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

Eksempel Tallet 42 er kvadratfritt, og $\varphi(42) = 12$. Dermed gir resultatet at vi har $a^{13} \equiv a \pmod{42}$ for alle heltall a. ♦

Oppgave La n være et kvadratfritt naturlig tall. Vis at da er $a^{l\cdot\varphi(n)+1} \equiv a \pmod{n}$ for alle $l \geqslant 0$. ♦

Løsning Vi viser dette ved induksjon på l. I basissteget har vi $l = 0$, og påstanden blir åpenbart sann. I induksjonssteget antar vi at påstanden holder for $l = k$, altså at $a^{k\cdot\varphi(n)+1} \equiv a \pmod{n}$. Da får vi:

$$\begin{aligned} a^{(k+1)\cdot\varphi(n)+1} &= a^{\varphi(n)} \cdot a^{k\cdot\varphi(n)+1} \\ &\equiv a^{\varphi(n)} \cdot a = a^{\varphi(n)+1} \\ &\equiv a \pmod{n}. \end{aligned}$$

Her brukte vi induksjonsantagelsen i overgangen fra første til andre linje, og resultat ovenfor helt til slutt. ♦

Eksempel Vi vil regne ut 18^{867} mod 42. Vi kan ikke bruke Eulers teorem, siden 18 og 42 ikke er relativt primiske. Men vi kan bruke at 42 er kvadratfri, og at $\varphi(42) = 12$, og få:

$$\begin{aligned} 18^{867} &= 18^{72\cdot 12+3} = 18^{72\cdot\varphi(42)+1} \cdot 18^2 \\ &\equiv 18 \cdot 18^2 = 18^3 = 5832 \equiv 36 \pmod{42}. \end{aligned}$$

Det vil si at $18^{867} \bmod 42 = 36$. ♦

Oppgave La n være kvadratfritt, og la e og d være naturlige tall som er hverandres inverser modulo $\varphi(n)$. Vis at da er $a^{ed} \equiv a \pmod{n}$ for alle heltall a. ♦

Løsning Vi har $ed \equiv 1 \pmod{\varphi(n)}$, så $ed = l \cdot \varphi(n) + 1$ for et heltall $l \geqslant 0$. Da får vi at $a^{ed} = a^{l\cdot\varphi(n)+1} \equiv a \pmod{n}$. ♦

Dette siste resultatet gir akkurat det vi trenger for å vise at RSA fungerer.

RSA: Fremgangsmåten

Vi starter med å formulere hvordan RSA fungerer, litt mer presist enn tidligere.

Nøkkelgenerering. Vi lager et RSA-nøkkelpar på følgende måte: Først velger vi to primtall p og q, med $p \neq q$, og lar $n = pq$. Så velger vi et tall e som er relativt primisk til $\varphi(n)$, og som oppfyller ulikhetene $0 < e < \varphi(n)$. Vi setter d til å være den inversen til e modulo $\varphi(n)$ som er slik at $0 < d < \varphi(n)$.

Den offentlige nøkkelen er (n, e), og den private nøkkelen er (p, q, d).

Kryptering. En melding m, som er et tall slik at $0 \leqslant m < n$, krypteres ved hjelp av funksjonen

$$f(m) = m^e \bmod n.$$

(Hvis vi vil kryptere en melding som er for stor til å representeres med et tall som er mindre enn n, må vi først dele den opp i mindre deler, og kryptere hver del for seg.)

Dekryptering. En kryptert melding c dekrypteres ved hjelp av funksjonen

$$g(c) = c^d \bmod n.$$

RSA: Korrekthet

Nå vil vi vise at dekrypteringen i RSA alltid gir riktig resultat, altså at hvis vi krypterer en melding m til en kryptert melding c, og deretter dekrypterer c, så får vi tilbake m. Med andre ord vil vi vise at likheten $g(f(m)) = m$ holder for alle meldinger m.

Vi har $f(m) \equiv m^e \pmod{n}$, og dermed får vi

$$g(f(m)) \equiv (m^e)^d = m^{ed} \equiv m \pmod{n}.$$

Siden $0 \leqslant m < n$ og $0 \leqslant g(f(m)) < n$, betyr dette at $g(f(m)) = m$.

Vi har dermed vist at dekrypteringen i RSA alltid gir riktig resultat.

RSA: Sikkerhet

Anta at Bob har offentlig nøkkel (n, e), og at Alice vil sende meldingen m til Bob. Hun krypterer den med Bobs offentlige nøkkel, og får den krypterte meldingen $c = m^e \bmod n$, som hun sender.

Eva, som avlytter all kommunikasjonen mellom Alice og Bob, ser tallet c, og ønsker å finne m. Vi kan dessuten regne med at hun kjenner til Bobs offentlige nøkkel (n, e), siden denne ikke er hemmelig.

Evas problem er altså: Gitt tall n, e og c, løs likningen

$$x^e \equiv c \pmod{n}.$$

Hvis Eva klarer å faktorisere n, så kan hun finne dekrypteringseksponenten i Bobs private nøkkel, og bruke den til å løse problemet.

Men det kan vel også tenkes at det er mulig å løse Evas problem på en annen måte, uten å måtte faktorisere n? Ja, men det finnes ingen kjent metode for å løse Evas problem som er enklere enn å faktorisere n.

Derfor regner vi RSA-systemet som sikkert, forutsatt at nøkkelen lages slik at det ikke er praktisk mulig å faktorisere n.

Bruk av tallteori i RSA

RSA-systemet er avhengig av mye av tallteorien vi har lært. Vi har sett at *korrektheten* i RSA (altså at dekryptering alltid gir den opprinnelige meldingen) bygger på Fermats lille teorem og Eulers teorem.

Videre er det flere teknikker som trengs for å gjøre RSA *praktisk brukbart*, altså for at det skal være mulig å regne ut alt som trengs:

- For å utføre kryptering og dekryptering trengs teknikken vi har lært for modulær potensregning: vi beregner $m^e \bmod n$ ved å først beregne $m^2 \bmod n$, $m^4 \bmod n$, $m^8 \bmod n$, og så videre.
- For å beregne $\varphi(n)$ trengs formelen vi har lært for φ.
- For å finne tallet d når vi lager nøkkelen trengs en metode for å beregne inverser. Vi har lært at vi kan bruke Euclids algoritme til dette.
- For å velge passende primtall p og q trengs en metode for primtallstesting, altså for å sjekke om et gitt tall er et primtall. Vi har lært én metode, basert på Fermats lille teorem.

Alle disse teknikkene fungerer godt for å utføre beregninger på en datamaskin, selv om modulusen n er veldig stor.

Til slutt er *sikkerheten* i RSA basert på at noen ting *ikke* er praktisk mulig å regne ut når tallene blir store, nemlig:

- Faktorisering av veldig store tall er ikke praktisk mulig.

Det som gjør at RSA fungerer og er sikkert, er at når tallene blir store, blir faktorisering enormt mye vanskeligere enn primtallstesting, modulær potensregning og så videre. Dermed er det mulig å få tallet n til å både være *lite nok* til at en vanlig datamaskin enkelt klarer å beregne alt som trengs i RSA og samtidig *stort nok* til at man ikke klarer å faktorisere det selv med de kraftigste datamaskinene som finnes.

Digresjon

En ny type datamaskin er i dag under konstruksjon, kjent som en *kvantedatamaskin*. Disse maskinene er fundamentalt annerledes enn vanlige datamaskiner og supercomputere, da de utnytter direkte naturlovene som styrer de minste partiklene vi kjenner til, nemlig kvantefysikkens lover, til å gjøre beregninger som ikke lar seg gjøre på vanlige datamaskiner. I stedet for en vanlig bit, som alltid er enten 0 eller 1, er kvantedatamaskiner bygget opp av såkalte kvantebits, eller *qubits*. Akkurat som Erwin Schrödinger i 1935 så for seg en katt som kunne være både levende og død på én gang, et tankeeksperiment vi i dag kjenner som *Schrödingers katt*, kan en kvantebit være i en såkalt *superposisjon* av 0 og 1 på én gang – helt til man observerer den, hvoretter den plutselig "bestemmer" seg for den éne eller den andre verdien.

Motivasjonen for å bygge slike kvantedatamaskiner var i utgangspunktet å simulere nettopp kvantefysikk, som så langt har motstått alle forsøk på effektiv simulering med vanlige datamaskiner. Ved siden av å være til hjelp i grunnforskning innen fysikk og kjemi, vil dette kunne åpne dørene for å designe nye materialer, stoffer, og medisiner via mer nøyaktige simuleringer av komplekse molekyler, så vel som å forstå for eksempel *kvantebiologiske* fenomener, som mistenkes å inkludere både fuglers evne til å navigere over store distanser, og selve grunnlaget for liv på jorden: fotosyntesen.

Slik var ståa i 1981, når man først begynte å diskutere om slike maskiner ville kunne la seg bygge. Men i 1994 sendte en matematiker ved navn Peter Shor sjokkbølger rundt fagmiljøet når han oppdaget at slike datamaskiner også kan gjøre noe helt spesielt, noe som ikke har noe med kvantefysikk å gjøre, og noe som har unnviket matematikere i hundrevis av år: de kan effektivt faktorisere et hvilket som helst tall, og beregne en hvilken som helst diskret logaritme. Som vi har sett innebærer dette at dersom en slik maskin ble bygget, ville den umiddelbart kunne brutt sikkerheten til både Diffie–Hellmans nøkkelutveksling og RSA!

Og dermed var kappløpet i gang: De neste tiårene så kvantedatamaskinen gå fra et snevert konsept spekulert over av et lite knippe forskere, til en storsatsing for flere av verdens mektigste nasjoner. Foreløpig burde vi dog kunne føle oss trygge: De største kvantedatamaskinene per i dag har et par hundre qubits, mens det er beregnet at flere millioner qubits vil trengs for å bryte dagens kryptografi. Men dette utvikler seg raskt, og det er ingen som vet nøyaktig når kvantedatamaskinene vil bli store nok til å utgjøre en sikkerhetstrussel. Samtidig fins det mye kryptert sensitiv data som skal hemmelighholdes på veldig lang sikt, fra pasientdata til statshemmeligheter.

Derfor trenger vi allerede nå ny *kvantesikker* asymmetrisk kryptografi. Det amerikanske organet NIST (National Institute of Standards and Technology) har de siste par årene organisert en prosess for å utvikle kvantesikre erstatninger for dagens kryptografi, og er nå i prosessen med å standardisere de første nye algoritmene. Disse er basert på nye enveisfunksjoner som, i motsetning til faktorisering og diskrete logaritmer, ikke lar seg reversere med en kvantedatamaskin så vidt vi vet. Men hvem vet, kanskje lusker det enda uoppdagede kvantealgoritmer i skyggene og bare venter på å bli oppdaget?