

Note omkring RSA kryptering.

Gert Læssøe Mikkelsen
Datalogisk institut
Aarhus Universitet

3. april 2009

1 Kryptering med offentlige nøgler

Indtil midt i 1970'erne troede næsten alle, der beskæftigede sig med kryptologi, at hvis en person Alice ønskede at sende en krypteret besked til en anden person Bob, så var Alice og Bob nødt til at mødes inden og udveksle en hemmelig nøgle. Dette lyder måske ikke som et stort problem, men i 1970'erne var eksempelvis banker begyndt at sende elektroniske beskeder til hinanden. Og det er nemt at se hvilke problemer, der i dag ville være med mange millioner brugere af Internettet, der ønsker at kunne lave sikre nethandler i mange tusinde netbutikker.

Midt i 70'erne finder Whitfield Diffie og Martin Hellman på en protokol¹, hvor Alice og Bob uden at dele nogen nøgle i forvejen og uden at mødes kan generere en hemmelig nøgle, der kan bruges til kryptering.

Diffie og Hellman viste, at den tidligere antagelse om, at Alice og Bob skal mødes og udveksle nøgler inden, de kan kommunikere. Deres arbejde inspirerer Ron Rivest, Adi Shamir og Len Adleman til at opfinde RSA kryptosystemet, som er det første kryptosystem med offentlige nøgler. Ideen bag offentlige nøgler er den følgende: Vi forestiller os at Bob får lavet et stort antal ens hængelåse, men kun en nøgle til dem. Bob sender nu hængelåse til alle posthuse. Når Alice vil sende en besked til Bob, putter hun beskeden i en kasse, går ned på posthuset og får udleveret en af Bobs hængelåse som hun smækker og låser kassen, derefter sender hun den låste kasse til Bob. Bob kan nu åbne kassen og læse beskeden, men da ingen andre end Bob har en nøgle, er det kun Bob, der kan læse beskeden.

For at dette skal kunne virke i praksis skal de fysiske hængelåse skiftes ud med en matematisk funktion en såkaldt "envejsfunktion" altså en funktion som det er let at udføre (lukke en hængelås), men hvor det er svært at beregne den inverse funktion (åbne hængelåsen) med mindre man har noget ekstra information (nøgler). I det følgende skal vi se at sådan en funktion kan laves på baggrund af talteori.

2 RSA kryptosystemet

En af de envejsfunktioner som RSA bygger på er *primtalsfaktorisering*. Det er meget nemt at gange to tal sammen, ved brug af en almindelig computer kan dette gøres på få millisekunder også selv om tallene indeholder over 250 cifre². Derimod antages det at det tager millioner af år for en supercomputer at udregne tallene p og q ud fra tallet N , hvis p og q er ca. ligestore primtal og $N = pq$. Nu har vi fundet noget der kunne bruges som den envejsfunktion, vi leder efter, vi skal bare have denne funktion hængt på beskeden, så denne bliver skjult.

Generering af nøgler Hvis Bob ønsker at lave et sæt RSA nøgler gøres det følgende:

¹En protokol er en beskrivelse af kommunikation mellem to parter

²Et tal med 250 cifre er meget stort - det antages at antallet af atomer i Universet er et tal med cirka 80 cifre

1. Vælg to meget store (over 250 cifre) primtal p og q
2. Beregn $N = pq$ og $\phi(N) = (p - 1)(q - 1)$
3. Vælg et heltal e så $0 < e < \phi(N)$ og sådan at e og N er indbyrdes primiske
4. Beregn d så $ed \equiv 1 \pmod{\phi(N)}$

Den offentlige nøgle er $\{N, e\}$ og den hemmelige nøgle er $\{N, d\}$.

Kryptering og dekryptering Hvis Alice vil sende en krypteret besked til Bob, og hun kender Bobs offentlige nøgle, kan hun gøre det på følgende måde:

Lad beskeden m være et heltal $< N$, cifferteksten c er nu givet ved følgende beregning:

$$c = m^e \pmod{N}$$

Bob kan dekryptere c med følgende beregning:

$$m = c^d \pmod{N}$$

Hvorfor virker RSA? Når Bob dekryptere beskeden med ovenstående beregning, virker det på grund af det følgende:

Først skal vi huske på at: $ed \equiv 1 \pmod{\phi(N)} \Rightarrow$ der eksisterer et heltal k så: $ed = 1 + k\phi(N)$

$$m \equiv c^d \tag{1}$$

$$\equiv (m^e)^d \tag{2}$$

$$\equiv m^{(ed)} \tag{3}$$

$$\equiv m^{(1+k\phi(N))} \tag{4}$$

$$\equiv (m^1)(m^{\phi(N)})^k \tag{5}$$

$$\equiv m1^k \equiv m \pmod{N} \tag{6}$$

Linie (5) til linie (6) er sand pga. Eulers sætning.

Sikkerheden af RSA Sikkerheden i RSA hviler på, at hvis man ikke kender p og q er det umådeligt svært at regne $\phi(N)$ ud og dermed beregne d . Og på at hvis man ikke kender m er det umådeligt svært at regne m ud givet $m^e \pmod{N}$, selv hvis man kender N og e . Det er klart at hvis primtalsfaktorisering er nemt, er det også nemt at regne $\phi(N)$ ud. Det at det er svært at regne m ud givet $m^e \pmod{N}$ kaldes for diskret logaritme og er et problem der også antages at være svært, på samme måde som primtalsfaktorisering.

3 Digitale signaturer

Indtil nu har vi beskæftiget os med kryptering, det vil sige det at sende en hemmelig besked, dette kaldes også for *konfidentialitet* - at holde noget hemmeligt. En anden lige så vigtig del af kryptologi er *authentifikation* - at bevise over for andre at man er den man påstår man er og at bevise at en besked kommer uændret frem.

Hvis vi kort overvejer hvordan ganske almindelige underskrifter (signaturer) virker, eller i hvert fald burde virke. Jeg kan underskrive et dokument hvilket gør at:

1. Det kan afgøres, at det er mig personligt, der har skrevet under.
2. Jeg er juridisk bundet af indholdet.
3. Modtageren af dokumentet kan vise det til en tredjepart, som kan verificere ovenstående.

Først forsøger vi at implementere den digitale version af underskrifter på følgende måde, der ligner den analoge udgave: Når Alice ønsker at underskrive et dokument vedhæfter hun en indskaning af hendes underskrift til dokumentet, og sender det afsted. Dette har oplagt den fejl at en anden person, lad os kalde hende Eva, kan kopiere underskriften og underskrive alle de dokumenter hun har lyst til.

Derfor er det meget vigtigt, at underskriften på en eller anden måde både hænger sammen med indholdet af dokumentet og vedkommende der skriver under. Her kommer RSA algoritmerne os endnu en gang til hjælp.

RSA signaturer Når Alice vil sende en besked til Bob, sådan at Bob kan være sikker på, at den kommer fra Alice og kan bevise dette overfor en tredje person, gør Alice det følgende:

1. Alice krypterer beskeden m med sin *hemmelige* nøgle så signaturen $\sigma = m^d \pmod N$
2. Alice sender beskeden m og signaturen σ til Bob
3. Bob accepterer signaturen, hvis: $m = \sigma^e \pmod N$

Det er nu klart at hvis en anden person har ændret beskeden uden at kende til d så vil Bob ikke godtage signaturen.

4 Hemmelighedsdeling og sikre flere-partsberegninger

I et tidligere afsnit så vi på kryptering som giver konfidentialitet, altså hemmeligholdelse mellem to parter. Dette giver os at Alice kan sende en besked til Bob uden at Eve kan opsnappe den og læse beskeden. En anden form for konfidentialitet er den

følgende: Alice har en hemmelighed, eksempelvis hendes dankort pinkode, som hun gerne vil opbevare hemmeligt. Alice kan give koden til Bob, og hvis hun senere glemmer den, kan hun få den igen fra Bob. Denne løsning har det oplagte problem, at Bob lærer hele hemmeligheden (pinkoden), heldigvis kender Alice til kryptologi så i stedet for at give hele hemmeligheden til Bob, deler hun den mellem Bob og Claire på følgende måde: Alice finder på et tilfældigt tal h_1 og udregner et andet tal h_2 sådan at $h_2 = pin - h_1$. Nu giver hun h_1 til Bob og h_2 til Claire. Hvis Alice senere glemmer pinkoden, kan hun få de to tal h_1 og h_2 fra Bob og Claire og udregne pinkoden igen, men hverken Bob eller Claire kender til pinkoden. Disse dele af hemmeligheden kalder vi *shares*.

Det ovenstående kan udvides til et system der tager højde for at de personer Alice giver dele af hemmeligheden til, måske også kan glemme disse dele. Dette foregår på følgende måde: Alice vælger en tilfældig linie sådan, at den skærer y-aksen i hemmeligheden, nu er $share_1 =$ værdien for linien i $x = 1$, $share_2 =$ værdien for linien i $x = 2$ og så videre. Hvis man kun har en share (et punkt), er der uendeligt mange linier og derfor er hemmeligheden stadig hemmelig. Hvis man har to shares (punkter) er linien entydigt givet, og man kan regne hemmeligheden ud. Dette kan udvides yderligere, så man i stedet for linier bruger polynomier (2. grads, 3. grads osv. ligninger), dette bygger på følgende fact: et polynomium af grad t er entydigt givet af $t + 1$ punkter. Dette kaldes *Shamir secret sharing* (samme Shamir som fra RSA).

Det er selvfølgelig meget smart at kunne gemme hemmeligheder, men der er en endnu mere smart ting ved overstående. Man kan regne på hemmeligheder, hvilket kaldes *sikre flere-partsberegninger*. Hvis Alice, Bob og Claire hver har et hemmeligt tal, og de gerne vil udregne summen uden at oplyse tallene, kan det gøres på følgende måde: De deler hver deres hemmelighed op i tre dele og giver en til hver. Nu lægger de hver deres tre shares sammen, og deres resultater er nu shares af resultatet af hemmelighederne (Se opgave 5). Det er bevist, at alle beregninger, der kan udføres af en part, også kan udregnes som sikre flere-partsberegninger.

5 Opgaver

Opgave 1: Eksempel på RSA kryptering Dette er et eksempel med alt for små tal til at systemet er sikkert. Brug følgende tal:

$p = 3$, $q = 11$, $d = 7$ og $e = 3$. Beskederne kan så være tal i intervallet 0 til 32.

1. Beregn N og $\phi(N)$ og vis at nøgleparret opfylder kravene til en nøgle.
2. Prøv at kryptere og dekryptere nogle værdier.

Opgave 2: Usikker brug af sikkert kryptosystem Alice har hørt at det er meget svært at bryde RSA kryptosystemet, derfor vælger hun at bruge RSA til at sende en besked til Bob. Hun gør det på følgende måde:

- Bogstaverne nummereres så $A = 0, B = 1, \dots, \mathring{A} = 28$.
- Hvert enkelt bogstav i beskeden krypteres hvert for sig med Bobs offentlige nøgle, og de krypterede bogstaver sendes til Bob.
- Bob kan med sin hemmelige nøgle dekryptere de modtagne bogstaver, og får beskeden.

Hvorfor er dette ikke sikkert ligegyldigt hvor mange bit der er i nøglen?

(Hint: Hvordan vil beskeden “hejmeddig” komme til at se ud i krypteret tilstand, (tænk på foredraget omkring historiske kryptosystemer))

Opgave 3: Deling af den hemmelige nøgle Tallene der skal bruges til RSA i praksis er så store at de på ingen måde kan huskes i hovedet. Man er derfor nødt til at opbevare den hemmelige nøgle et eller andet sted, og der er derfor en vis risiko for at den kan blive stjålet.

Vi skal nu se nærmere på hvordan man kan reducere risikoen for at det sker, ved at dele den hemmelige nøgle op i to dele. Det gør man ved at vælge to tal d_1, d_2 helt tilfældigt, dog sådan at

$$d_1 + d_2 = d \quad (7)$$

Nu opbevares disse to tal forskellige steder, f.eks. to forskellige computere. I eksemplet ovenfor kunne vi f.eks. have $d_1 = 17$ og $d_2 = -10$.

3.1) Argumenter for, at hvis nogen får fat i d_1 eller d_2 , men ikke dem begge, så har vedkommende stadig ingen anelse om hvad d er.

Ideen er nu at lave en underskrift på m ved, at den computer der har d_1 beregner $m^{d_1} \pmod N$, tilsvarende beregnes $m^{d_2} \pmod N$ af den computer der kender d_2 . Begge “halve signaturer” kan nu samles:

3.2) Vis at man kan beregne signaturen $m^d \pmod N$ ud fra de to bidrag $m^{d_1} \pmod N$ og $m^{d_2} \pmod N$, nemlig sådan her:

$$m^d \pmod N = (m^{d_1} \pmod N)(m^{d_2} \pmod N) \pmod N$$

Opgave 4: Beregning af store potenser Når man skal regne på de meget store tal der i virkeligheden bruges til RSA, er det vigtigt at bruge de mest effektive metoder. Selvom computere er hurtige, går det alligevel alt for langsomt hvis vi ikke tænker os om når vi programmerer dem. Et eksempel: Forestil Jer at I skal beregne $23^{17} \pmod{33}$ med blyant og papir eller med en lommeregner, der kun kan de grundlæggende regningsarter. Hvis man bare går i gang uden at tænke, ville man måske gøre som antydnet her:

$$23^{17} \pmod{33} =$$

$$23 \cdot 23 \cdot 23 \cdot 23 \cdot 23 \cdot 23 \cdot 23 \cdot 23 \cdot 23 \cdot 23 \cdot 23 \cdot 23 \cdot 23 \cdot 23 \cdot 23 \cdot 23 \pmod{33}$$

altså bare gange 23 med sig selv 17 gange. Det virker som en temmelig stor opgave, men er heldigvis helt unødvendigt.

4.1) Argumenter for at

$$\begin{aligned}23^{17} \bmod 33 &= (23^{16} \bmod 33) \cdot 23 \bmod 33; \\23^{16} \bmod 33 &= (23^8 \bmod 33) \cdot (23^8 \bmod 33) \bmod 33; \\23^8 \bmod 33 &= (23^4 \bmod 33) \cdot (23^4 \bmod 33) \bmod 33; \\&\dots \text{(fortsæt selv her)}\end{aligned}$$

Brug dette til at forklare hvordan $23^{17} \bmod 33$ kan beregnes med kun 5 multiplikationer og divisioner.

Opgaven her er et eksempel på en generel teknik der hedder “square and multiply”, som bruges i alle computerprogrammer der anvender RSA. Den er en helt nødvendig forudsætning for at RSA kan bruges til noget i praksis. For de skarpe knive i skuffen kommer her en opgave der går tættere på denne metode.

Fact: ethvert positivt tal kan skrives som en sum af 2-potenser (1, 2, 4, 8, etc.):

$$1 = 1; 2 = 2; 3 = 1 + 2; 4 = 4; 5 = 1 + 4; 6 = 4 + 2; 7 = 4 + 2 + 1; \dots \quad (8)$$

Dem der kender til binære tal ved hvorfor, men det behøver man ikke at vide noget om i denne opgave.

4.2) Brug ovenstående fact og ideen i Opgave 4.1 til at designe en metode der beregner $a^e \bmod N$ for vilkårlige positive tal a, e, N . Hvor mange multiplikationer og divisioner skal man bruge med Jeres metode, hvis $e = 1026$? Bemærk at den naive metode bruger 1025 multiplikationer.

Opgave 5: Hemmelighedsdeling og sikre flere-partsberegninger Disse opgaver viser, hvordan man regner på delte hemmeligheder.

5.1) Lav et eksempel, der viser, at hvis der er tre hemmelige tal 5, 3 og 8 så kan $5 + 3 - 8$ udregnes sikkert.

(Lav en linie for hvert tal, lav shares, og hvis at disse kan man regne på)

5.2) Lav et eksempel, der viser, at hvis der er et hemmeligt tal 11 så kan det kendte tal 5 ganges med hemmeligheden så 5×11 kan udregnes.

(Lav en linie for hemmeligheden, og hvis at det kan lade sig gøre at gange det kendte tal på)

5.3) Lav et eksempel der viser, at det er sværere at gange 2 hemmelige tal sammen. (Problemmet er det følgende: hvis man ganger to 1. grads polynomier (linier) sammen er resultatet et 2. grads polynomie (en parabel), der findes dog løsninger på dette, disse kræver, at parterne skal kommunikere sammen for at få graden ned igen.)