Institut for Matematik og Datalogi
Syddansk Universitet, Odense

27. november 2018
RF

# Algoritmiske aspekter af RSA

## Eksponentiering

Til RSA bruges tre positive heltal: $n$, $e$, og $d$. Beskeden, som skal krypteres, er et heltal $m < n$. Ligeledes er beskedens krypterede version et heltal $c < n$.

- Kryptering: beregn $c = m^e \pmod{n}$
- Dekryptering: beregn $m = c^d \pmod{n}$

Der er altså brug for effektivt at kunne eksponentiere, dvs. beregne $x^k$ for positive heltal $x$ og $k$. En naiv metode er at gange $x$ sammen $k$ gange: $x^k = x \cdot x \cdot x \cdot x \ldots x$. Dette bruger $k - 1$ multiplikationer. Flg. metode til at beregne $x^k$ er langt hurtigere:

- Hvis $k = 0$, svar med 1 [da $x^0 = 1$ altid].
- Hvis $k \geq 1$ og $k$ er lige: beregn $y = x^{k/2}$ og svar med $y \cdot y$ [da dette er lig $x^{k/2} \cdot x^{k/2} = x^{k/2+k/2} = x^k$].
- Hvis $k \geq 1$ og $k$ er ulige: beregn $y = x^{(k-1)/2}$ og svar med $x \cdot y \cdot y$ [da dette er lig $x \cdot x^{(k-1)/2} \cdot x^{(k-1)/2} = x^{1+(k-1)/2+(k-1)/2} = x^k$].

Dette er en rekursiv algoritme, som i hvert rekursivt kald arbejder med samme $x$, men med en eksponent som er blevet mindst halveret. Derfor foretages der højst $\log k$ rekursive kald før eksponenten når under 1. Da hvert kald i sig selv udfører højst 2 multiplikationer, bliver det samlede antal multiplikationer højst $2 \log k$.

I RSAs tilfælde er $x$ og $k$ tal med 1024 bits eller mere ($x$ er $m$ og $c$ ovenfor, og $k$ er $e$ og $d$, alle heltal af størrelse op til $n$, som typisk har dette antal bits). Hvis $x$ og $k$ har 1024 bits, vil $x^2$ have $2 \cdot 1024$ bits, $x^3$ have $3 \cdot 1024$ bits, etc.,

og $x^k$ vil have $k \cdot 1024 \approx 2^{1024} \cdot 1024 = 2^{1024} \cdot 2^{10} = 2^{1034} \approx 10^{311}$ bits. Det vil kræve tæt på $10^{301}$ Gb RAM bare at gemme dette tal i hukommelsen!

I RSA ikke skal man dog heldigvis ikke bruge $x^k$, men $x^k \pmod n$, et tal der ikke har flere bits end $n$. For at beregne dette uden at konstruere tal med for mange bits undervejs, kan man bruge flg. simple faktum (kendt fra DM549 Diskrete Metoder til Datalogi, se *Rosen* side 256):

$$a \cdot b \pmod n \quad = \quad (a \pmod n) \cdot (b \pmod n) \pmod n$$

Dette betyder, at hvis vi bruger ovenstående rekursive algoritme, men altid undervejs med det samme *efter enhver multiplikation* erstatter det beregnede tal med dets værdi modulus $n$, så ender vi med samme resultat modulus $n$ i sidste ende. Med andre ord kan vi beregne $x^k \pmod n$ ved følgende rekursive algoritme:

- Hvis $k = 0$, svar med 1.
- Hvis $k \geq 1$ og $k$ er lige: beregn $y = x^{k/2} \pmod n$ og svar med $y \cdot y \pmod n$.
- Hvis $k \geq 1$ og $k$ er ulige: beregn $y = x^{(k-1)/2} \pmod n$ og svar med $x \cdot (y \cdot y \pmod n) \pmod n$.

Nu har alle tal konstrueret undervejs ca. samme antal bits som $n$.

## Finde talværdier til RSA

Til RSA bruges tre positive heltal: $n$, $e$, og $d$. Metoderne bag at finde disse beskrives nu:

- Tallet $n$ findes som produktet af to primtal $p$ og $q$, dvs. $n = pq$. Der findes effektive metoder (f.eks. Rabin-Miller-testen) til at teste, hvorvidt et givet tal er et primtal. Da man kan vise, at primtal ikke er specielt sjældne (primtalssætningen, der siger at antallet af primtal mindre end $x$ er omtrent $x/\ln x$), finder man $p$ og $q$ ved at vælge tilfældige tal, og teste dem for, om de er primtal. Man stopper, når to primtal er fundet (hvilket forventet vil tage omtrent $\ln x$ forsøg). Hvis man gerne vil have $n$ til at bestå af 1024 bits, vælger man tilfældige

tal med ca. 512 bits. Når $p$ og $q$ er fundet, vil $n = pq$ så have ca. $512 + 512 = 1024$ bits.

- Tallet $e$ skal vælges, så $\gcd(e, n') = 1$ for $n' = (p-1)(q-1)$. Da Euclids algoritme (se afsnit 4.3.7 side 284–285 i *Rosen*) effektivt kan beregne $\gcd(e, n')$, findes $e$ ved at kigge på tilfældige tal $e$ med det rette antal bits, og teste dem for om $\gcd(e, n') = 1$, indtil man finder et, der opfylder dette. Sådanne tal kan vises ikke at være specielt sjældne – alle primtal $e$ vil f.eks. opfylde det ønskede, medmindre $n'$ er et multiplum af $e$. Det sidste er meget usandsynligt for tilfældige, store tal $e$, så man rammer ca. lige så nemt et sådant brugbar tal $e$ som man rammer et primtal.

- Tallet $d$ skal vælges, så $de = 1 \pmod{n'}$, for $n' = (p-1)(q-1)$. Her udnyttes at $\gcd(e, n') = 1$, hvilket betyder (se afsnit 4.3.8 side 285–287 i *Rosen*) at der findes heltal $s$ og $t$ sådan at $1 = se + tn'$. For disse gælder, regnet modulus $n'$, at $1 = se + tn' = se = (s \pmod{n'})e$, så det ønskede $d$ vælges som $s \pmod{n'}$. Da $s$ og $t$ kan beregnes med den udvidede udgave af Euclids algoritme, kan $d$ findes effektivt, når $e$ og $n'$ kendes. Én version af den udvidede udgave af Euclids algoritme finder $s$ og $t$ ved at arbejde sig baglæns gennem beregningerne lavet at Euclids algoritme (forklaret med et eksempel i *Rosen* side 286). En anden version af samme algoritme finder $s$ og $t$ undervejs i Euclids algoritme (forklaret med et eksempel i *Rosen* side 287). På slides i DM534 er den anden version brugt.

## Korrekthed af RSA

Korrektheden af RSA, dvs. at dekryptering og kryptering er hinandens inverse (at $c^d = (m^e)^d = m \pmod{n}$) når $n$, $e$ og $d$ er valgt som angivet ovenfor, blev vist til forelæsningen (og kan genfindes på side 317–318 i *Rosen*). Ingredienserne er Fermats lille sætning og den kinesiske restklassesætning. Korrekthedsbeviset er ikke emnet for noterne her.

Note: Alle referencer til *Rosen* er til 8. udgave (Global Edition) af *Discrete Mathematics and Its Applications*, Kenneth H. Rosen, 2018.