Studieretningsprojekt

2019/2020

Opgaveformulering:

Redegør for RSA kryptering og gennemgå relevante resultater der viser at RSA-kryptering er sikker og at det er muligt at kryptere og dekryptere.

Vis hvordan det er muligt at bryde RSA krypterede data såfremt krypterede data er tilgængelig.

Skriv et program til kryptering og dekryptering af data. Dit program skal kommenteres.

Skriv et program til dekryptering af data hvor den private nøgle ikke er kendt. Skal ligeledes kommenteres.

Sebastian Borch Andersen

SR-fag: Matematik A, 2. Fag: Informatik B

Vejledere: Dennis Pipenbring (DPI) og Jakob Hermann (JHE)

23. marts 2020

Resumé

Formålet med opgaven er, at redegøre for RSA-kryptering og at vise, at det med RSA-kryptering er muligt at kryptere og dekryptere, samt at det er sikkert og hvordan det er muligt at bryde. Derudover at lave et program til kryptering og dekryptering af data, og et program til at bryde RSA-kryptering. For at vise at RSA-kryptering er muligt at kryptere og dekryptere, gennemgår jeg den talteori, der er relevant i forhold til, at oprette nøglerne i RSA-kryptering og til hvorfor $c = m^e \pmod{n}$ kan bruges til kryptering og $m = c^d \pmod{n}$ til dekryptering, når nøglerne er bestemt ud fra:

1) Vælg to primtal q og p, og sæt $n = pq$
2) Beregn $\varphi(n) = (p-1)(q-1)$
3) Vælg et helt tal e, hvor at $0 < e < \varphi(n)$ og $(e,\varphi(n)) = 1$
4) Til sidst beregnes tallet d, så $ed \equiv 1 \pmod{n}$

Hvor de offentlige nøgler er n og e, og den private nøgle er d. Jeg viser også, hvordan man kan bruge en metode som Fermats metode, til at bryde RSA-kryptereingen. Fermats metode bygger på, at n kun kan faktoriseres, hvis og kun hvis der findes heltallige løsninger, x og y, til ligningen $n = x^2 - y^2$. Derudover viser jeg at RSA-kryptering er sikkert, da det tager lang tid at bryde RSA-kryptering. Til sidst kommenterer jeg på et de programmer, jeg har lavet. Som konklusion, er det altså med RSA-kryptering muligt at kryptere og dekryptere, og er n stor nok vil krypteringen være sikker.

Indhold

1 Indledning

Jeg vil i denne opgave redegøre for RSA-kryptering. Det vil jeg gøre ved at gennemgå relevant talteori i forhold til RSA-kryptering, samt at gennemgå relevante resultater der viser, at det er muligt at kryptere og dekryptere RSA-kryptering. Derudover vil jeg vise hvordan det er muligt at bryde RSA-kryptering, ud fra de offentlige nøgler. Til dette vil jeg gennemgå Fermats metode. Dertil vil jeg også vise at RSA-kryptering er sikkert. Til sidst vil jeg skrive et program, der kan RSA kryptere og dekryptere i PHP. Samt skrive et program, der kan dekryptere RSA-kryptering ved at finde den private/hemmelige nøgle, dette skrives ligeledes i PHP. Begge programmer vil jeg kommentere på.

2 Talteori

Talteori omhandler egenskaber ved de hele tal $\mathbb{Z}$. Altså addition, subtraktion, multiplikation og division. Her er division den mere problematiske, da man ikke nødvendigvis får hele tal ved division. Dette afsnit vil bruges på at forklare relevante egenskaber og definitioner i forhold til RSA-kryptering.

2.1 Divisibilitet og regning med rester

Et helt tal $a \neq 0$ er *divisor* i et helt tal b, hvis der findes et helt tal q, så:

$$b = qa \Rightarrow \frac{b}{a} = q \tag{1}$$

Det kan også skrives som $a|b$ og tallet q kaldes *kvotienten.*

For tre hele tal a, b og c gælder det at:

$$c|a \wedge c|b \Rightarrow c|(xa + yb) \text{ for } x,y \in \mathbb{Z}, c \neq 0 \tag{2}$$

Det gælder, da der findes q_1 og $q_2 \in \mathbb{Z}$, så $a = q_1 c$ og $b = q_2 c$. Ved indsættelse af x og y giver det $xa + yb = xq_1 c + yq_2 c = (xq_1 + yq_2)c$, hvilket er $c|(xa + yb)$.

Man kan også regne division med rest, hvor q bestemmes sådan, at det er en rest, r, som er et helt tal. Hvor det for tallene, n og m, gælder at hele

tal q og r kan bestemmes sådan at:

$$m = qn + r, \quad 0 \leq r < n \tag{3}$$

Ved division af m med n, vil man altså få $q + r$. Den principale rest, r, er defineret som:

$$m \pmod{n} = \text{den principale rest af } m \text{ ved division med } n \tag{4}$$

Det siges m *modulo* n. Der er nogle simple egenskaber ved $m \pmod{n}$, en af dem er:

$$m \pmod{n} = m, \quad 0 \leq m < n \tag{5}$$

Det kommer som følger af (3), hvor m kan skrives på formen $m = 0 \cdot n + m$. En anden egenskab er:

$$(m + k \cdot n) \pmod{n} = m \pmod{n} \tag{6}$$

For at bevise det, sættes $r = m \pmod{n}$. Nu kan m skrives som $m = qn + r$. kn adderes på begge sider af $m = qn + r$. Dette giver $m + kn = (q + k)n + r$ som kan omskrives til $m = qn + r$, hvilket stemmer overens med (3). Denne egenskab kan bruges til at forklare

$$(a \cdot b) \pmod{n} = (a \pmod{n} \cdot b \pmod{n}) \pmod{n} \tag{7}$$

og

$$a^t \pmod{n} = (a \pmod{n})^t \pmod{n} \tag{8}$$

Det gøres ved brug af (3), der siger, at der findes hele tal q_1, q_2 og r_1, r_2 sådan at $a = q_1 n + r_1$, $0 \leq r_1 < n$ og $b = q_2 n + r_2$, $0 \leq r_2 < n$. Dette kan der nu regnes på:

$$\begin{aligned} (a \cdot b) \pmod{n} &= ((q_1 n + r_1) \cdot (q_2 n + r_2)) \pmod{n} \\ &= (r_1 r_2 + (q_1 q_2 n + q_1 r_2 + q_2 r_1)n) \pmod{n} \end{aligned} \tag{9}$$

Herfra kan (6) bruges, sådan at:

$$\begin{aligned} (a \cdot b) \pmod{n} &= r_1 r_2 \pmod{n} \\ &= (a \pmod{n} \cdot b \pmod{n}) \pmod{n} \end{aligned} \tag{10}$$

Det samme gælder for (8), hvis der i stedet skrives $a \cdot a$. Hvilket kan skrives som a^2 eller a^t, hvor t er antallet af a.

2.2 Fælles divisorer

For to hele tal a og b, hvor de begge ikke er 0, kan der findes fælles divisorer, d, hvis $d|a$ og $d|b$. To tal forskellig fra 0, har et endeligt antal fælles divisorer, derfor vil der også være en største fælles divisor, som betegnes (a,b). For de to hele tal a og b, som ikke er 0, gælder det at:

$$(a,b) \geq 1 \wedge (a,b) = (b,a) \wedge (-a,b) = (a,b) \tag{11}$$

Det gælder også at

$$(a,b) = (a,r) = (a,b \pmod n) \tag{12}$$

hvis q og r er valgt sådan at $b = qa + r,\ 0 \leq r < a$. Dermed er $r = b - qa$. Herefter sættes $d = (a,b)$ og $d_1 = (a,b - qa)$. Da $d|a$ og $d|b$ kan man ud fra (2) sige, at $d|(b - qa)$. Altså at d er en fælles divisor i a og $b - qa$. Da $d_1|a$ og $d_1|(b - qa)$ følger det af (2), at $d_1|(b - qa) + qa$, hvilket medfører at $d_1|b$. d_1 er altså også en fælles divisor i a og b, derfor er $d = d_1$ og $(a,b) = (a,r)$. Da det gælder at $(a,b) = (b,a)$ vil man også kunne skrive:

$$(a,b) = (r,b) = (a \pmod n,b) \tag{13}$$

For at bestemme den største fælles divisor, kan ovenstående bruges. Det gøres ved at finde resten ved division af det største tal med det mindste tal. Hvilket gentages indtil man får 0, hvor det andet tal vil være den største fælles divisor mellem start tallene. Det kan skrives op på følgende måde:

$$(a,b) = (r_1,b) = (r_1,r_2),(r_3,r_2) = ... = (r_j,0) = r_j \tag{14}$$

For to hele tal, a og b, som ikke er 0, vil der findes hele tal s og t, så:

$$(a,b) = sa + tb \tag{15}$$

Kigger vi på (14) ses det, at $r_1 = a - q_1 b = 1 \cdot a + (-q_1)b$. Sættes $s_1 = 1$ og $t_1 = -q_1$, vil vi få $r_1 = s_1 \cdot a + t_1 b$. Mere generelt kan det skrives $r_k = s_k a + t_k b$. For a, b, $n \in \mathbb{Z}$ gælder det at:

$$(a,n) = 1 \ \wedge \ (b,n) = 1 \Rightarrow (ab,n) = 1 \tag{16}$$

Ifølge (15) findes der s_1, t_1 så $s_1a + t_1n = 1$ og s_2, t_2 så $s_2b + t_2n = 1$. Ved multiplikation af de to fås:

$$1 = (s_1a + t_1n)(s_2b + t_2n) = (s_1s_2)ab + (t_1s_2b + s_1t_2a + t_1t_2n)n \qquad (17)$$

Skrives s_1s_2 som s og $t_1s_2b + s_1t_2a + t_1t_2n$ som t, får vi $sab + tn$. Af (15) kan vi derfor skrive $(ab,n) = 1$.

2.3 Primtal

Primtal defineres som, et helt tal større end 1, hvor kun 1 og tallet selv er divisorer i tallet. Der findes uendeligt mange primtal, ligesom der findes uendeligt mange tal. De tal der ikke er primtal kaldes sammensatte tal.

2.4 Kongruens

Har man a, b, $n \in \mathbb{Z}$, $n > 0$, gælder det at a er *kongruent med b modulo n*, hvis $n|(a - b)$. Dette skrives $a \equiv b \pmod{n}$. b kaldes en rest af a *modulo n*. Det gør det fordi, der for $n|(a - b)$ vil kunne findes et tal $q \in \mathbb{Z}$, sådan at $a - b = qn$. Dette kan omsrives til $a = b + qn$, hvilket viser, at b vil være en rest ved division af a med n.
Det gælder også at:

$$a \equiv b \pmod{n} \Leftrightarrow a \pmod{n} = b \pmod{n} \qquad (18)$$

Igen findes der et tal, $q \in \mathbb{Z}$, sådan at $a = b + qn$, hvilket giver $a \pmod{n} = (b + qn) \pmod{n}$. Ud fra (6) kan det så skives som $b \pmod{n}$, hvilket vil sige, at $a \pmod{n} = b \pmod{n}$.
Man kan også gå en anden vej og sige, at hvis $a \pmod{n} = b \pmod{n} = r$, så vil der være hele tal q_1 og q_2, sådan at $a = q_1n + r$ og $b = q_2n + r$. Ved subtraktion fås $a - b = (q_1 - q_2)$n, hvilket svarer til $n|(a - b)$, og derfor også $a \equiv b \pmod{n}$.

Det er muligt at dividere en kongruens på begge sider, hvis $(a,n) = 1$

$$ax \equiv ay \pmod{n} \Rightarrow x \equiv y \pmod{n} \tag{19}$$

$ax \equiv ay \pmod{n}$ vil per defintion være $n|(ax - ay)$ og dermed $n|a(x - y)$. Idet $(a,n) = 1$ vil det gælde at $n|an$, altså er n fælles divisor i $a(x - y)$ og an, hvilket vil sige, at $n|(x - y)$ og dermed $x \equiv y \pmod{n}$.
For Eulers φ-funktion som kaldes φ, gælder det for et primtal, p, at $\varphi(p) = p - 1$. For to primtal p og q, vil det så være:

$$\varphi(pq) = (p - 1)(q - 1) \tag{20}$$

Gælder det at $(a,n) = 1$, så er

$$a^{\varphi(n)} \equiv 1 \pmod{n} \tag{21}$$

Sættes $r_i \in \mathbb{Z}_n^* = \{r_1,r_2,...,r_{\varphi(n)}\}$, så er $(r_i,n) = 1$. Da $(a,n) = 1$ vil vi som følger af (16) også få at $(ar_i,n) = 1$. Ifølge (12) kan det derfor skrives $(ar_i \pmod{n},n) = (ar_i,n) = 1$. Da $ar_i \pmod{n} < n$, følger det at $ar_i \pmod{n} \in \mathbb{Z}_n^*$. Nu har vi, at tallene $ar_1 \pmod{n}$, $ar_2 \pmod{n}$, ..., $ar_{\varphi(n)} \pmod{n}$ er de samme som r_1, r_2, ..., $r_{\varphi(n)}$, de er dog ikke nødvendigvis i samme rækkefølge. Derfor kan vi skrive:

$$\begin{aligned} &r_1 \cdot r_2 \cdot ... \cdot r_{\varphi(n)} \pmod{n} \\ &\quad = ((ar_1 \pmod{n}) \cdot (ar_2 \pmod{n}) \cdot ... \cdot (ar_{\varphi(n)} \pmod{n})) \pmod{n} \end{aligned} \tag{22}$$

Af (7) kan det nu skrives:

$$\begin{aligned} r_1 \cdot r_2 \cdot ... \cdot r_{\varphi(n)} \pmod{n} &= ar_1 \cdot ar_2 \cdot ... \cdot ar_{\varphi(n)} \pmod{n} \\ &= a^{\varphi(n)} \cdot r_1 \cdot r_2 \cdot ... \cdot r_{\varphi(n)} \pmod{n} \end{aligned} \tag{23}$$

Da $(r_i,n) = 1$, kan man som følger af (19) forkorte r_i væk. Gøres dette er kun $a^{\varphi(n)} \pmod{n} = 1 \pmod{n}$ tilbage, som er det samme som $a^{\varphi(n)} \equiv 1 \pmod{n}$ ifølge (18).
Er det stadig gældende at $(a,n) = 1$, vil det for alle $a \in \mathbb{Z}$ og $s \in \mathbb{N}$ gælde:

$$a^s \pmod{n} = a^{s \pmod{\varphi(n)}} \pmod{n} \tag{24}$$

Ifølge (3) vil der kunne findes to hele tal q og r, så $s = q \cdot \varphi + r$, $0 \leq r < \varphi(n)$. Det vil sige at (24) nu kan skrives som

$$a^s \pmod{n} = a^{q \cdot \varphi(n) + r} \pmod{n} = (a^{q \cdot \varphi(n)} \cdot a^r) \pmod{n} \tag{25}$$

Af egenskaberne (7) og (8) kan det nu skrives

$$a^s \pmod{n} = (((a^{\varphi(n)} \pmod{n})^q \pmod{n}) \cdot a^r \pmod{n}) \pmod{n} \tag{26}$$

Da $a^{\varphi(n)} \pmod{n}$=1 (mod n), kan det nu skrives som, ved brug af (8)

$$a^s \pmod{n} = ((1^q \pmod{n}) \cdot (a^r \pmod{n})) \pmod{n} = a^r \pmod{n} \tag{27}$$

Da r er resten ved division af s med $\varphi(n)$ er $r = s \pmod{\varphi(n)}$. Heraf fås $a^s \pmod{n} = a^{s \pmod{\varphi(n)}} \pmod{n}$.

3 RSA-kryptering

RSA-kryptering er udviklet af Ron Rivest, Adi Shamir og Len Adleman i 1977. Systemet er baseret på primfaktoriseringsproblemet, hvor sikkerheden også ligger. Før man kan kryptere og dekryptere, skal man først oprette to offentlige nøgler, n og e, og en hemmelig nøgle, d. De udregnes ved følgende trin:

1) Vælg to primtal q og p, og sæt $n = pq$
2) Beregn $\varphi(n) = (p-1)(q-1)$
3) Vælg et helt tal e, hvor at $0 < e < \varphi(n)$ og $(e,\varphi(n)) = 1$
4) Til sidst beregnes tallet d, så $ed \equiv 1 \pmod{\varphi(n)}$

Når man så skal kryptere en tekst, skal den først omskrives til tal og inddeles i blokke. Hver blok bliver repræsenteret som tallet m, hvor $0 < m < n$. Til enkrypteringen af m bruges følgende:

$$c = m^e \pmod{n} \tag{28}$$

Hvor c er hver blok, m, enkrypteret. For at dekryptere c, bruges:

$$m = c^d \pmod{n} \tag{29}$$

Grunden til at det virker, kan findes ved skrive (28) og (29) sammen og udnytte (8):

$$m = (m^e \pmod{n})^d \pmod{n} = m^{ed} \pmod{n} \tag{30}$$

Bruges (24) kan det nu skrives som:

$$m = m^{ed \pmod{\varphi(n)}} \pmod{n} \tag{31}$$

Når nøglerne bestemmes, bestemmes d således at $ed \equiv 1 (\mathrm{mod}\ \varphi(n))$. Ifølge (18) kan det også skrives som $ed \pmod{\varphi(n)} = 1$, hvilket kan indsættes så:

$$m = m^1 \pmod{n} \Rightarrow m = m \pmod{n} \tag{32}$$

Hvilket stemmer overens med (5). Disse regler kan kun følges, fordi det gælder at $(e,\varphi(n)) = 1$.

3.1 Eksempel

Til at starte med skal nøglerne først bestemmes, for at gøre det vælges to primtal $p = 97$ og $q = 151$. Derefter bestemmes $n = 97 \cdot 151 = 14647$. Nu kan $\varphi(n)$ beregnes:

$$\varphi(n) = (p-1)(q-1) = 14400 \tag{33}$$

Nu vælges e, sådan at $0 < e < \varphi(n)$ og $(e,\varphi(n)) = 1$. Der er mange muligheder for e, men da der kun skal bruges en vælges $e = 7$. Til sidst beregnes d, således at $ed \equiv 1 \pmod{n}$, hvilket giver at $d = 12343$. De offentlige nøgler er altså $n = 14647$ og $e = 7$, og den hemmelige nøgle er $d = 12343$.

Disse nøgler kan nu bruges til at kryptere en klartekst. Klarteksten vælges til "hello world". Hvert bogstav laves nu om til numre på to cifre. Til det, bruges den normale nummerering af alfabetet : a = 01, b = 02, ..., z = 26. For at få mellemrum med, skrives mellemrum i dette eksempel som tallet 53. Teksten, som nu er tal, deles op i blokke på 4 cifre, hvilket er det samme som to bogstaver. Dette giver så:

he	ll	o␣	wo	rl	d␣
0805	1212	1553	2315	1812	0453

Blok størrelsen på 4, vælges fordi det tal den enkelte blok repræsenterer, skal være mindre n. Det sidste mellemrum tilføjes, sådan at alle blokke er på 4 cifre. Nu kan hver blok enkrypteres ved at bruge (28):

$$\begin{aligned} m \to m^e \pmod{n} &= c \\ 0805 \to 0805^7 \pmod{14647} &= 14587 \\ 1212 \to 1212^7 \pmod{14647} &= 4002 \\ 1553 \to 1553^7 \pmod{14647} &= 14551 \\ 2315 \to 2315^7 \pmod{14647} &= 3111 \\ 1812 \to 1812^7 \pmod{14647} &= 11929 \\ 0453 \to 0453^7 \pmod{14647} &= 302 \end{aligned} \tag{34}$$

For at dekryptere beskeden skal vi bruge den hemmelige nøgle d og (29):

$$\begin{aligned} c &\to c^d \pmod{n} = m \\ 14587 &\to 14587^{12343} \pmod{14647} = 805 \\ 4002 &\to 4002^{12343} \pmod{14647} = 1212 \\ 14551 &\to 14551^{12343} \pmod{14647} = 1553 \\ 3111 &\to 3111^{12343} \pmod{14647} = 2315 \\ 11929 &\to 11929^{12343} \pmod{14647} = 1812 \\ 302 &\to 302^{12343} \pmod{14647} = 453 \end{aligned} \tag{35}$$

De samme resultater kommer tilbage igen, altså virker RSA-krypteringen. Nogle af resultaterne giver et tre cifret resultalt, her skal man blot tilføje et 0 foran resultatet, så det stemmer overens med bogstaverne for det gældende tal. Når man har gjort det, vil man kunne lave tallene om til bogstaver, og man står igen med "hello world".

3.2 Brydning og sikkerhed

For at bryde RSA-krypteringen kræver det, at man finder de to primtal q og p, eller ved at finde $\varphi(n)$. Da $\varphi(n)$ udregnes på baggrund af q og p, giver det mest mening at finde dem. Man kunne også forstille sig, at man kunne finde m direkte fra $m^e \pmod{n}$. Det er dog svært at udregne m ud fra $m^e \pmod{n}$, selvom man kender e og n, derfor giver det heller ikke mening at bryde krypteringen den vej. Det logiske er derfor at kigge på, hvordan man finder de to primtal. Det kan gøres ud fra n, da det er produktet af p og q, og da n er en af de offentlige nøgler, kender vi allerede n.
For at finde p og q kan man faktorisere n. En måde at gøre dette på er Fermats metode. Fermats metode bygger på at n er et ulige helt tal, og at n kun kan faktoriseres, hvis der til ligningen $n = x^2 - y^2$ findes løsninger, x og y, som er hele tal. Er der heltallige løsninger til $n = x^2 - y^2$, kan det også skrives som:

$$n = x^2 - y^2 = (x + y)(x - y) \tag{36}$$

Kan n faktoriseres som $n = p \cdot q$, og p er størst, kan n også skrives:

$$n = p \cdot q = \left(\frac{p+q}{2}\right)^2 - \left(\frac{p-q}{2}\right)^2 \tag{37}$$

Fordi n er ulige, må både p og q være ulige, og da $p \geq q$ er $\frac{p+q}{2}$ og $\frac{p-q}{2}$ positive hele tal. For at finde x og y, omskrives (36):

$$x^2 - n = y^2 \tag{38}$$

Nu vælges k sådan at $k^2 > n$, da y^2 ikke kan være negativ. Herefter køres en løkke igennem, hvor $x = (k + j)$ og j starter på 0, indtil $x^2 - n$ giver et kvadrattal. Giver det et kvadrattal, er der altså fundet heltallige løsninger x og y. De kan nu bruges til at finde p og q:

$$n = (x+y)(x-y) = p \cdot q \Rightarrow p = (x+y) \wedge q = (x-y) \tag{39}$$

Da man nu kender p og q, kan man finde $\varphi(n)$ og derefter den hemmelige nøgle d. Altså er krypteringen brudt. Fermats metode virker bedst, når p og q er næsten lige store. Så når man skal vælge primtal til RSA-krypteringen, kan man med fordel gardere sig mod denne metode, ved at vælge primtal der ikke er næsten lige store.

Selvom det er muligt at faktorisere n, er RSA-kryptering stadig sikkert. Det skyldes at nøglen, n, er et enormt stort tal. Har n 154 cifre[1], vil faktoriseringen kræver mindst $5{,}5 \cdot 10^{19}$ operationer. En moderne supercomputer[2], har 148.600 TFlops/s, hvilket svarer til at den kan klare $148600 \cdot 10^{12}$ operationer pr. sekund. Det kan nu regnes lidt på:

$$\frac{5{,}5 \cdot 10^{19} \text{ operationer}}{148600 \cdot 10^{12} \text{ operationer pr. sek}} \approx 370 \text{ sek} = 6 \text{ min og } 10 \text{ sek} \tag{40}$$

Dette er rimelig hurtigt, og ikke særligt sikkert. Selvom det er de færreste der har en supercomuter, og specielt en supercomputer der er så hurtig, er det for nemt at dekryptere. For at gøre det svære at dekryptere, kan man

[1] [1] KRYPTOLOGI: s. 119
[2] [2] TOP500: Rank 1

gøre nøglen, n, større. Er n på 308 cifre[3], vil det kræve mindst $3{,}8 \cdot 10^{29}$ operationer. Igen kan det regnes på det:

$$\frac{3{,}8 \cdot 10^{29} \text{ operationer}}{148600 \cdot 10^{12} \text{ operationer pr. sek}} = 2.557200538 \cdot 10^{12} \text{ sek} \approx 80000 \text{ år} \quad (41)$$

Dette er meget længere tid, og efter 80000 år vil den krypteret data, efterhånden være lidt ligegyldig. Når det tager så lang tid, må man altså kunne sige, at det er svært at faktorisere, og krypteringen er derfor sikker. For at nedsætte tiden, kan man enten bygge en hurtigere computer, eller udvikle hurtigere algoritmer. Indtil videre er der ikke fundet en algoritme, der er så hurtig, at RSA-kryptering ikke længere er sikker. Så længe man ikke gør det, vil RSA-kryptering være sikkert, også selvom computerne bliver hurtigere. Når computerne bliver hurtigere vil man nemlig også være i stand til at gøre nøglerne større, og svære at faktorisere.

[3] [1] KRYPTOLOGI: s. 119

4 RSA-program

Følgende program bestemmer primtallene og nøglerne, der skal bruges til RSA-kryptering. Når nøglerne er oprettet, kan programmet kryptere og dekryptere en besked.

```
<?php

// Functionen tjek_primtal() tjekker om et givet tal er et primtal.

function tjek_primtal($tal){
        if (($tal == 1) or ($tal == 0)) {
                return false;

// Er tallet 1 eller 0, er det ikke et primtal, derfor sendes false tilbage.

        }

// Tallet testes for, om det er et primtal, ved at tjekke om nogen tal
går op i det givet tal. Der er ikke nogen grund til at teste 1, da 1
går op i alle tal, derfor starter $i med at være 2. Derudover er der
ikke nogen tal over halvdelen af det givet tal, der vil gå op i det givet
tal, der for testes kun $tal/2 og under.

        for ($i = 2; $i <= ($tal / 2); $i++) {
                if($tal % $i == 0) {
                        return false;

// Hvis et tal går op i det givet tal, er det ikke et primtal, og false
returneres.

                }
        }
        return true;

// Er tallet over 1, og ingen andre tal går op i tallet, må det være
et primtal. true returneres derfor.

}
```

```
// primtal() finder alle primtal, mellem $min og $max, og sætter dem
i en liste.
function primtal($min,$max){
        $tal=$min;
        $prim=[];
        while($tal<=$max){
// tjek_primtal() bruges til at teste, om det er et primtal, er det tilfældet
sættes primtallet i listen $prim.
                if(tjek_primtal($tal)===true){
                        $prim[]=$tal;
                }
                $tal++;
        }
        return $prim;

}
// bestem_e() bestemmer e, sådan at (e,$k)=1
function bestem_e($k,$min){
// Følgende laver en liste over alle primtal, mindre end det interval
der bruges til at bestemme primtallene. 2 springes over, da $k er et
lige tal.
        $prim=primtal(3,$min);
// Nedstående tjekker om ($prim[$i],$k)=1. Er det tilfældet er e fundet,
og det returneres.
        while(sfd($prim[$i],$k)!=1){
                        $i++;
                }

        return $prim[$i];

}
```

```
// sfd() finder største fælles divisor

function sfd($a, $b)
{

// Er enten $a eller $b lig med 0, er den modsatte den største
   fælles divisor. Ifølge (14)

    if ($a == 0){
       return $b;
     }
    if ($b == 0){
       return $a;
    }

// Er $a og $b ens, er den største fælles divisor både $a og $b.

    if($a == $b){
        return $a ;
     }

// Finder resten af det største tal, indtil $a eller $b er 0.
   Metoden følger af (14).

    if($a > $b) {
        return sfd( $a-$b , $b ) ;
    }
    return sfd( $a , $b-$a ) ;
}

// find_d() finder d, sådan at ed ≡ 1 (mod n).

function find_d($e,$k){
        $q=0;
        do{

// Da ed ≡ 1 (mod φ(n)) også kan skrives som d = (φ(n) · q + 1)/e. Der
ændres på q i stedet for d, da dette vil være hurtigere. Resultatet
afrundes, da q sjældent er et helt tal. φ(n) er $k.

                $d=round(($k*$q+1)/$e);
                $q++;
```

```
        }
// Tjekker om ed (mod φ(n)) = 1 er opfyldt. Er det opfyldt er d fundet.

        while(bcmod(bcmod($e,$k)*bcmod($d,$k),$k)!=1);

        return $d;

}
// opretnogler() opretter de nøgler, der skal bruges til krypteringen.

function opretnogler($min,$max){
// Først findes primtallene. For at gøre det laves en liste med primtal
mellem $min og $max, ved brug af primtal(). Herefter vælges to tilfældige
primtal, der ikke er ens.
        $prim=primtal($min,$max);
        $p=$q=$prim[rand(0,count($prim)-1)];
        while($q==$p){
                $p=$prim[rand(0,count($prim)-1)];
        }
// n og φ(n) beregnes som $n og $k.
        $n = $p*$q;
        $k = ($p-1)*($q-1);
// e er $e, og bestemmes ud fra $k, og $min sikrer at $e er mindre end
$k.
        $e=bestem_e($k,$min);
// Ud fra $e og $k findes $d, som er den hemmelige nøgle.
        $d=find_d($e,$k);
// Nøglerne er nu bestemt og sendes tilbage i en liste.
        return array($d, $e, $n);
}
```

```
// krypter() krypterer den besked, $bes, der skal krypteres.
function krypter($bes,$min,$max){
// Den oprindelige besked gemmes i variablen $besked.
        $besked = $bes;
// Længden af beskeden findes, og er længden ulige tilføjes et mellemrum.

        $bes_l = strlen($bes);
        if(1==$bes_l%2){
                $bes.= "␣";
        }
// Nøglerne oprettes, hvor primtallene bestemmes mellem $min og $max.

        $nogler = opretnogler($min,$max);
        $d=$nogler[0];
        $e=$nogler[1];
        $n=$nogler[2];
// Tegn omskrives til tal.
    $tegn = array("a", "b", "c", "d", "e", "f", "g", "h",
    "i", "j", "k", "l", "m", "n", "o", "p", "q", "r", "s",
    "t", "u", "v", "w", "x", "y", "å", "A", "B", "C", "D",
    "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O",
    "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z",
    "␣", ".", ",", ":", "-", "/");
    $tal = array("01", "02", "03", "04", "05", "06", "07",
    "08", "09", "10", "11", "12", "13", "14", "15", "16",
    "17", "18", "19", "20", "21", "22", "23", "24", "25",
    "26", "27", "28", "29", "30", "31", "32", "33", "34",
    "35", "36", "37", "38", "39", "40", "41", "42", "43",
    "44", "45", "46", "47", "48", "49", "50", "51", "52",
    "53", "54", "55", "56", "57", "58");

        $bes = str_replace($tegn, $tal, $bes);
```

```
// Tallene deles op i blokke af fire cifre.
    $bes = str_split($bes,4);
// Hver blok krypteres nu, og adskilles med mellemrum.
    for($i=0; $i < count($bes); $i++){
        $c=bcmod(bcpow($bes[$i],$e),$n);
        $k_bes .= $c." ";
    }
// Den oprindelige besked, den krypteret besked og nøglerne sendes tilbage.

    return array($besked, $k_bes, $d, $e, $n);


}
// dekrypter() dekrypterer en besked, $bes, ved hjælpe af nøglerne $d
og $n.
function dekrypter($bes, $d, $n){
// Tallene splittes op i deres blokke i en liste.

        $bes = explode(" ", $bes);
// Hver blok dekrypteres
        for($i=0; $i < count($bes); $i++){

                $m = bcmod(bcpow($bes[$i],$d),$n);
// Er tallet under 1000 tilføjes et 0 foran tallet, for at sikre hver
blok er på fire cifre og at det kan skrives om til tegn igen.
                if($m < 1000) {
                        $m = "0".$m;
                }
```

```
// Fjerner overflødigt 00.
                    if($m == "00") {
                            $m = NULL;
                    }
// Tallene splittes op, så hvert tal på to cifre, kan omskrives til tegn.

                    $m = str_split($m, 2);
                    $dek .= $m[0]."␣";
                    $dek .= $m[1]."␣";
            }

$tegn = array("a", "b", "c", "d", "e", "f", "g", "h",
"i", "j", "k", "l", "m", "n", "o", "p", "q", "r", "s",
"t", "u", "v", "w", "x", "y", "z", "A", "B", "C", "D",
"E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O",
"P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z",
"␣", ".", ",", ":", "-", "/");
$tal = array("01␣", "02␣", "03␣", "04␣", "05␣", "06␣",
"07␣", "08␣", "09␣", "10␣", "11␣", "12␣", "13␣", "14␣",
"15␣", "16␣", "17␣", "18␣", "19␣", "20␣", "21␣", "22␣",
"23␣", "24␣", "25␣", "26␣", "27␣", "28␣", "29␣", "30␣",
"31␣", "32␣", "33␣", "34␣", "35␣", "36␣", "37␣", "38␣",
"39␣", "40␣", "41␣", "42␣", "43␣", "44␣", "45␣", "46␣",
"47␣", "48␣", "49␣", "50␣", "51␣", "52␣", "53␣", "54␣",
"55␣", "56␣", "57␣", "58␣");
    $dek = str_replace($tal, $tegn, $dek);
// Den dekrypteret besked kan nu sendes tilbage.
 return $dek;
}

?>
```

5 Program til dekryptering uden nøglen, d

Følgende program finder primtallene, der er brugt til at skabe nøglerne, ud fra n. Når primtallene er bestemt, finder programmet den private nøgle d.

```
<?php
// find_prim() finder den hemmelige nøgle d, samt primtallene der er brugt. Den bygger på Fermats metode. Funktionen skal bruge de offentlige nøgler.
    function find_prim($n,$e){
// $k findes sådan at $k^2 er det laveste hele tal større end $n.
        $k=round(sqrt($n));
        $i=0;
// Følgende finder den værdi, hvor $x^2−$n giver et kvadrattal. For at teste om det giver et kvadrattal, tages kvadratroden af $yy og afrundes, derefter tages kvadratet af dette. Kvadratet af det afrundet tal, vil kun give $yy, hvis tallet er et kvadrattal.
        do{
                $yy=bcpow($k+$i,2)-$n;
                $x=$k+$i;
                $i++;
        }while(bcpow(round(sqrt($yy)),2)!=$yy);
// $y udregnes, og ud fra $y og $x kan primtallene $q og $p findes. $k eller φ(n) kan også udregnes nu.
        $y=sqrt($yy);
        $p=$x+$y;
        $q=$x-$y;
        $k = ($p-1)*($q-1);
// Den hemmelige nøgle kan findes, ved brug af funktionen find_d(), da man nu kender både e og k, der skrives $e og $k.
        $d=find_d($e,$k);
```

```
        return array($p,$q,$d);
}

// Da nøglerne nu kendes kan funktion dekrypter() bruges, til at dekryptere
en besked.

?>
```

6 Konklusion

Med RSA-kryptering er det altså muligt at kryptere data, samt at dekryptere, når den private nøgle er kendt. Er den private nøgle ikke kendt, er det noget sværere at dekryptere data. Selvom det er muligt, med en metode som Fermats metode at bryde RSA-kryptering, vil det være tilstrækkeligt svært at bryde, hvis de valgte primtal er store nok. Da der ikke findes en god metode til at primfaktorisere, vil det for selv de hurtigste computere, tage alt for lang tid at dekryptere. Så lang tid, at dataet vil være irrelevant. Det at det tager lang tid at dekryptere RSA-kryptering, er hvad der gør RSA-kryptering sikkert. Så længe man ikke finder en hurtig måde at primfaktorisere, vil RSA-kryptering være sikkert. Selv når computer hastighederne stiger, da vil man blot kunne øge længden af primtallene og dermed n.

Litteratur

[1] Landrock, Peter. & Nissen, Knud. (1997). Kryptologi: - fra viden til videnskab. Abacus.

[2] TOP500 - november 2019, https://www.top500.org/lists/2019/11/ besøgt: 18-03 2020

[3] Riber, Peter. (2007). KRYPTERING. SYSTIME.

[4] Landrock, Peter. & Nissen, Knud. (1992). Kryptologi. Abacus.