TEC
Teknisk Erhvervsskole Center

14. DEC 2014

RSA-KRYPTERING

Studieretningsprojekt

Blerim Cazimi

Frederiksberg Tekniske Gymnasium

Matematik A

Vejleder: Jonas Kromann Olden

Informationsteknologi B

Vejleder: Kenneth Hebel

Indhold

Problemformulering

- Giv en kort redegørelse for hvordan kryptering og dekryptering foregår ved hjælp af RSA metoden.
- Gør rede for den matematiske teori om der ligger til grund for metodens sikkerhed.
- Giv en grundig redegørelse for teorien bag Pierre de Fermat's metode til faktorisering af tal. Lav et gennemregnet eksempel der demonstrerer hvordan metoden virker.
- Undersøg sammenhængen mellem tallets størrelse i bits og den mængde beregninger - eventuelt udtrykt i tid - der behøves for at faktorisere tallet i primtal.
- Kommenter den historiske udvikling i størrelsen af de tal der bruges som nøgler i forbindelse med RSA kryptering
- Vurder fremtidsperspektiverne for kryptering med RSA metoden.

Indledning

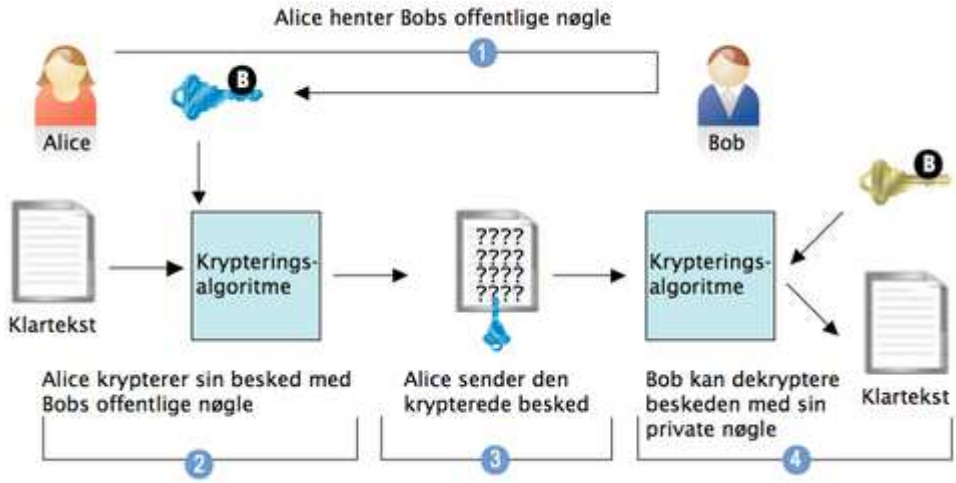
I et informationssamfund spiller kryptering en vigtig rolle i sikkerheden af fortrolige oplysninger. Kryptering sørger for, at dine fortrolige oplysninger er ulæselige for tredjeparter, hvor kryptering fx bruges hver gang, når du som kunde, skal købe varer fra nettet, eller når du blot skal tjekke din mail. Kryptering er derfor et vigtigt element i vores digitale samfund, hvor krypteringen sørger for, at vi alle blandt et informationssamfund kan føle os trygge over, at tredjeparter ikke kan få adgang til vores fortrolige oplysninger, men det kan være et spørgsmål om tid, før det måske bliver muligt for tredjeparter at finde en måde på, at få adgang til dine fortrolige oplysninger.

I denne rapport redegøres der for RSA-kryptosystemet, hvorefter de grundlæggende matematiske strukturer bag RSA-kryptosystemet beskrives. Derefter vil der blive vist et matematisk eksempel på, hvordan kryptering og dekryptering foregår med RSA. Dernæst vil der blive kigget på den matematiske teori bag metodens sikkerhed. Afslutningsvis, vil fremtidsperspektiverne blive vurderet for RSA-metoden. Derfor ligger hovedfokus i denne rapport på RSA-kryptosystemet.

Fordi word ikke tillader, at bruge fodnoter i en tekstboks, vil der under tekstboksene stå *Fodnote x* med henvisning til, hvor viden stammer fra.

RSA et asymmetrisk krypteringssystem

RSA er et asymmetrisk krypteringssystem og bliver anvendt til kryptering og digital signatur. Asymmetrisk kryptering er et krypteringssystem hvor afsender og modtager har hver sit sæt nøgler – en hemmelig privat nøgle, som kun er kendt af ejermanden, og en offentlig nøgle som alle har adgang til. Data, som krypteres med den offentlige nøgle, kan kun dekrypteres med den private nøgle. Lad os antage, at vi har 2 personer som ønsker at kommunikere fortroligt, Bob og Alice. Alice ønsker at sende en besked til Bob, derfor bruger hun Bobs offentlige nøgle til at kryptere beskeden. Alice sender nu den krypterede besked til Bob, som kun Bob kan dekryptere, fordi det er kun ham som kender sin private nøgle. Hvis Bob vil svare på Alices besked, så er proceduren den samme som før. Bob henter Alices offentlige nøgle, som han krypterer sit svar med, og sender den krypterede besked til Alice. Når Alice har modtaget beskeden, bruger hun sin private nøgle til at dekryptere Bobs besked. Dermed hviler sikkerheden på disse nøgler, hvor tredjeparter ikke kan dekryptere en besked, hvis ikke den private nøgle kendes. Det var disse tanker de tre herre Ron **R**ivest, Adi **S**hamir og Len **A**dleman havde, hvor de gik ind i jagten på at udvikle et asymmetrisk krypteringssystem, hvor det i 1977 lykkedes dem at løse og beskrive den ovennævnte situation matematisk, hvor de skabte det første asymmetriske krypteringssystem kaldet **RSA**[1].



Figur 1 - Asymmetrisk kryptering med privat og offentlig nøgle. Kilde: se fodnote 1

[1] Damhus, Martin, m.fl. *Informationsteknologi* kapitel 9.5.4

RSA-kryptering i praksis

RSA-kryptering er en envejsfunktion, hvor det er nemt for Bob at kryptere beskeden, men umuligt for andre end Alice at dekryptere beskeden uden kendskab til Alices private nøgle. Matematisk set er der tale om primfaktorisering, hvor det er nemt, at beregne produktet af to tal, men tidskrævende at gå den anden vej og faktorisere produktet. Det er det, der gør RSA sikkert, at det er tidskrævende at primfaktorisere produktet. Det er derfor muligt for andre end ejermanden af den private nøgle, at dekryptere chifferteksten (*den krypterede tekst*), hvis de kan beregne sig frem til den private nøgle, men da den private nøgle er et produkt af primtal på flere hundrede cifre der benyttes praktisk, så tager det længere tid for en super computer, at primfaktorisere produktet, end for solen at brænde ud[2]. Dette er hvad man kalder for faktoringsproblemet, som baserer sig på, at det er tidskrævende, at finde et stort tals primfaktorer, men nemt at beregne produktet af primfaktorerne[3].

Talteori

Talteori er en vigtig gren af matematikken, der beskæftiger sig mest tallene, specielt de hele tal, $\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$, og de naturlige tal[4], $\mathbb{N} = \mathbb{Z}^+ = \{1, 2, 3, \dots\}$. Der gennemgås de vigtige emner, der har betydning for RSA-kryptosystemet. Det skal bemærkes, at når der siges, at et tal skal være ikke-negativt heltal, betyder det, at tallet skal være et element i $\{0, 1, 2, 3, \dots\}$.

Regning med rester

En vigtig del af talteorien, som er nødvendig for forståelsen af RSA, er regningen med rest. Før vi kommer til det, skal vi først kigge på heltalsdivision Der introduceres en definition generelt om delelighed nedenfor:

Definition 1 (Heltalsdivision): Et naturligt tal a kaldes divisor i et heltal b, hvis der findes et heltal kvotient q, således at $b = qa$. Vi betegner denne med $a|b$, hvilket vil sige, at a går op i b eller, at b er deleligt med a.

Det er værd at bemærke, at der ofte ved division forekommer en rest. Dette kommer vi til at se på senere. Først skal vi kigge på følgende sætning:

[2] Matlex (?) http://goo.gl/z3FVWsl RSA - systemet: Lokaliseret 01/12/14
[3] Nielsen, (2004) http://goo.gl/sJvkB4 Sikkert og pålideligt peer-to-peer filsystem, side. 17. Lokaliseret 04/12/14
[4] Redaktionen (2009) http://goo.gl/e3rmzR Talteori: Lokaliseret 04/12/14

Sætning 2: Der gælder, for ethvert hele tal a, b og c:

① Hvis $a|b$ så vil $a \mid bc$, for $a \neq 0$.

② $ac|bc \Leftrightarrow a|b$, for $a, c \neq 0$.

③ Hvis a|b og b|c så vil $a|c$, for $a, c \neq 0$.

④ Hvis $c|a$ og c|b så vil $c|(xa + yb)$, for de hele tal x og y, og $c \neq 0$.

Bevis:

① Da a går op i b, findes der et tal $q \in \mathbb{Z}$, således at $b = qa$. Ganges c på hver side, får vi $bc = qca$. Her kan qc betragtes som ny kvotient, hvormed vi konkluderer, at a faktisk går op i bc i denne situation.

② Vi ser på implikationen "$\Longrightarrow$": Da ac går op i bc, findes der et $q \in \mathbb{Z}$, således at $bq = (bc)/(ac)$. Ved at forkorte dette med c, får vi $q = b/a$, som ønsket. Dvs. vi har vist, at $ac|bc \Longrightarrow a|b$. Vi ser på implikationen "$\Longleftarrow$": Da a går op i b, findes der et $q \in \mathbb{Z}$, således at $q = b/a$. Ved at forlænge dette med c, får vi $q = (bc)/(ac)$, som ønsket. Dvs. vi har vist, at $ac|bc \Longleftarrow a|b$.

③ Da a går op i b og b går op i c, findes der hhv. et $q_1 \in \mathbb{Z}$ og et $q_2 \in \mathbb{Z}$, således at $b = q_1 a$ og $c = q_2 b$. Ved at indsætte denne ene ligning i den anden, får vi $c = q_2(q_1 a) = (q_1 q_2)a$. Her kan $q_1 q_2$ betragtes som ny kvotient, hvormed vi konkluderer, at a går op i c i denne situation.

④ Da c går op i både a og b, findes der hhv. et $q_1 \in \mathbb{Z}$ og et $q_2 \in \mathbb{Z}$, således at $a = q_1 c$ og $b = q_2 c$. Da er $xa + yb = x(q_1 c) + y(q_2 c) = (xq_1 + yq_2)c$, hvilket svarer til, at c går op i $xa + yb$ i denne situation, som giver $xq_1 + yq_2 \in \mathbb{Z}$ som en ny kvotient.

Definition 3: Et naturligt tal divisor a i de hele tal b og c, siges at være en fælles divisor for a og b. Den største fælles divisor for tallene a og b betegnes normalt $\gcd(a, b)$. Hvis $\gcd(a, b) = 1$, siges a og b at være indbyrdes primiske.

Fodnote[5]

[5] Vestergaard, http://goo.gl/HF28DX RSA-kryptosystemet, side. 8. Lokaliseret 06/12/14

Udtrykket gcd er en forkortelse på engelsk for **G**reatest **C**ommon **D**ivisor. Vi vil, for simpelhedens skyld, fremover benytte notationen (a, b) frem for $\gcd(a, b)$.

Der ses nedenfor, at de mulige tal med dens rest kan gå op i 76 følgende:

$$76 = (8 \cdot 11) - 12 = (7 \cdot 10) + 6 = (12 \cdot 5) + 16.$$

Det vil sige, at hvis 76 divideres med 8, fås kvotienten 11 og en rest -12, eller hvis med 7, får vi kvotienten 10 og en rest 6, eller hvis med 12, får vi kvotienten 5 med rest 16. Vi vil i dette tilfælde fokusere på en positiv rest, der skal være mindre end divisoren. Denne rest kaldes for den principale rest.

Sætning 4: For alle naturligt tal a og heltal b, findes der unikke heltal q og r, således at $b = qa + r$, for $r \in [0, a[$.

Et unikt tal vil sige, at tallet er entydigt bestemt, dvs. der er kun ét tal, der opfylder en bestemt ligning.

Fodnote[6]

Euklids algoritme[7]

Med Euklids algoritme kan vi bestemme den største fælles divisor for to naturlige tal $a \geq b$, således at vi kan finde værdien (a, b). Vi starter med at se på følgende definition:

Definition 5: Der gælder $(a, b) = (a, r)$, hvor r er resten af b ved division med a.

Eksempel:

For at vise Euklids algoritme er ideen blot, at anvende division algoritmen gentagne gange indtil vi når den før sidste rest bliver nul. Vi har følgende:

$$a = bq_1 + r_1, \text{ dvs. } (a, b) = (b, r_1) \text{ for } r_1 \in [0, q_1[,$$

$$b = r_1 q_2 + r_2, \text{ dvs. } (b, r_1) = (r_1, r_2) \text{ for } r_2 \in [0, q_2[,$$

[6] Landrock, Peter, m.fl. *Kryptologi – fra viden til videnskab*. Side 71-73

[7] Courant Institute of Mathematical Sciences (2013) http://goo.gl/ARpPcz "Greatest common divisor" s. 3 Lokaliseret 09/12/14

$$r_1 = r_2q_3 + r_3, \text{ dvs. } (r_1, r_2) = (r_2, r_3) \text{ for } r_3 \in [0, q_3[,$$

… (fortsættes) …

$$r_{n-2} = r_{n-1}q_n + r_n, \text{ dvs. } (r_{n-2}, r_{n-1}) = (r_{n-1}, r_n) \text{ for } r_n \in [0, q_n[$$

$$r_{n-1} = r_nq_{n+1}, \text{ dvs. } (r_n, 0) = r_n$$

hvormed vi konkluderer, at

$$(a, b) = (b, r_1) = (r_1, r_2) = (r_2, r_3) = \ \ldots \ = (r_{n-2}, r_{n-1}) = (r_{n-1}, r_n) = (r_n, 0) = r_n.$$

Det vil sige, at det sidste ikke-negative tal rest r_n i Euklids algoritme for naturlige tal $a \geq b$ er deres største fælles divisor.

Eksempel: Vi finder den største fælles divisor i 29 og 17:

$$29 = (1 \cdot 17) + 12$$
$$17 = (1 \cdot 12) + 5$$
$$12 = (2 \cdot 5) + 2$$
$$5 = (2 \cdot 2) + 1$$
$$2 = (2 \cdot 1) + 0$$

altså er $(29, 17) = (17, 12) = (12, 5) = (5, 2) = (2, 1) = (1, 0) = 1$.

Da $(29, 17) = 1$, følger det af definition 3, at de to tal 29 og 17 er indbyrdes primiske. I bilag 5 er der vedhæftet et program der bruger euklids algoritme.

Kongruenser[8]

I dette afsnit indføres begrebet Modulo, som bruges i matematikkens verden i forbindelse med kongrens[9].

Definition 6: Lad a og b være heltal samt m være naturligt tal. Hvis m går op i $a - b$, dvs. $m|(a - b)$, siger vi, at a er kongruent med b modulo m. Vi betegner dette med $a \equiv b \pmod{m}$.

[8] Landrock, Peter, m.fl. *Kryptologi – fra viden til videnskab*. Side 87-88
[9] Rasmussen m.fl. (2013) http://goo.gl/l7SA6B, "modulo" Lokaliseret 11/12/14

Eksempel:

Da 6 går op i $32 - 2$, dvs. $6|(32 - 2)$, kan vi skrive $32 \equiv 2 \pmod 6$.

Dette kan også skrives på følgende måde:

$32 \equiv 2 \pmod 6 \equiv 2 \pmod 5 \equiv 2 \pmod 3 \equiv 2 \pmod 2 \equiv 2 \pmod 1$.

Primtal

Dette emne vil kun blive behandlet overordnet, da det antages, at du som læser allerede ved hvad et primtal er i forvejen. Vi vil trodsalt bevise følgende sætning:

Sætning 7: Antallet af primtal er endeløst.

Bevis: Vi beviser det ved modstrid. Antag, at der ikke findes et endeløst antal primtal, p_1, p_2, …., p_k. Betragt tallet $N = p_1 \cdot p_2 \cdot \ldots \cdot p_k+1$, hvor N er forskelligt fra de andre primfaktorer. Da ingen af dem går op i N (resten ved division er 1), må N være et primtal.

Fodnote[10]

Med denne viden, ved vi nu at der ikke er nogen grænser for, hvor store primtal der kan generes. Det er gode nyheder, da vi senere skal kigge på, at det er svært at primfaktorisere store tal.

Euler's φ-funktion[11]

I dette underafsnit redegøres der for Eulers φ-funktion. Funktionen skrives som φ(n), hvor n er et vilkårligt tal.

Som definition 3 siger**,** så er to tal indbyrdes primiske, hvis deres største fælles divisor er 1. Dette er brugbart fordi Eulers φ-funktion defineres ved antallet af hele tal, der er indbyrdes primisk med tallet n.

[10] RUC (2011) http://goo.gl/h8sli5 "Bevis teknikker" Lokaliseret 11/12/14

[11] Hansen, Johan, m.fl. Algebra og talteori, s. 42-43

Sætning 8

① For et primtal p gælder at:

$$\varphi(\mathrm{p}) = p - 1$$

② For to forskellige primtal p og q gælder at:

$$\varphi(\mathrm{p}) = (p-1)(q-1)$$

Bevis: Produktet af de to forskellige primtal kaldes for m, dvs. m = p · q. Antallet af indbyrdes primiske tal med m er antallet af tal der er større end 0 og mindre end m: (p · q) - 1 samt minus af alle de tal p og q går op i dvs.:

1p, 2p, 3p … q·p (de tal p går op i)

1q, 2q, 3q … p·q (de tal q går op i)

Dermed bringer det os til følgende udregning:

$$(pq-1) - (q-1) - (p-1) = (p-1)(q-1)$$

Tallene der går op i p og q trækkes fra alle tallene mellem 1 og tallet m.

Eksempel: Hvis vi har primtallene, $p = 5$ og $q = 3$, kan vi skrive:

$$\varphi(5 \cdot 3) = (5-1) \cdot (3-1) \Leftrightarrow \varphi(15) = 4 \cdot 2 \Leftrightarrow \varphi(15) = 8$$

Dermed er der 8 tal, som er indbyrdes primiske med tallet 15.

Inverse elementer[12]

Inverse-elementer vil blive brugbar i forbindelse med dekryptering af chifferteksten:

Sætning 9: Lad $\mathbb{F} \coloneqq \{0,1,\dots,n-1\}$, hvor n er et naturligt tal. Hvis der findes et x i $\mathbb{F}$, har vi det inverse element til a i $\mathbb{F}$ modulo n, således at $ax \equiv 1 \pmod{n}$. Med $a^{-1} \pmod{n}$ betegner vi det inverse element til a modulo n.

Eksempel:

$3 \cdot 8 = 1 \bmod 24$. Det inverse til 3 modulo er 8.

$8 \cdot 3 = 1 \bmod 24$. Det inverse til 8 modulo er 3.

Her ser vi, at 3 og 8 er inverse modulo til hinanden.

Sætning 10: Der findes et unikt inverst element til a modulo n, hvis og kun hvis $(a,n) = 1$.

Faktoriseringsteori

I dette afsnit gennemgås Fermats faktoriserings algoritme, som senere skal bruges til at analysere sammenhængen mellem bits og den tid der skal bruges for at faktorisere tallet.

Fermats metode[13]

Pierre de Fermat var en fransk jurist og amatørmatematiker. Han opfandt i 1643 en algoritme til at faktorisere både små- og store tal. Fermats metode bygger på følgende sætning:

Sætning 11

Lad n være et ulige helt tal. Tallet kan faktoriseres, hvis og kun hvis der er lige heltal løsninger til ligningen $n = x^2 - y^2$ mht. x og y.

[12] Landrock, Peter, m.fl. *Kryptologi – fra viden til videnskab*. Side 94-95
[13] Landrock, Peter, m.fl. *Kryptologi – fra viden til videnskab*. Side 129-130

Bevis

"Kun hvis": Antag at $n = x^2 - y^2$ har lige heltal løsninger. Ved anvendelse af 3. kvadratsætning, har vi $n = x^2 - y^2 = (x - y)(x + y)$, hvilket indikerer, at både $x - y$ og $x + y$ er ulige heltal, idet n er det.

"Hvis": Lad $n = n_1 n_2$ for $n_1 \leq n_2$. Bemærk, at n_1 og n_2 er ulige heltal, i det n er det. Lad $x = \frac{1}{2}(n_1 + n_2)$ og $y = \frac{1}{2}(n_2 - n_1)$. Bemærk, at x og y er heltal. Ved løsningen af to ligninger med to ubekendte, får vi $n_1 = x - y$ og $n_2 = x + y$. Dermed fås $n = n_1 \cdot n_2 = (x - y)(x + y) = x^2 - y^2$

Denne sætning giver os en ide til at danne en faktoriserings algoritme. Omskriver vi ligningen, ved at trække med $n - y^2$ på hver side, får vi $y^2 = x^2 - n$. Lad k være det mindste heltal, opfyldende $k^2 > n$, og betragt følgen:

$$\{k^2 - n, (k+1)^2 - n, (k+2)^2 - n, \dots, (k+j)^2 - n, \dots\}.$$

for alle ikke-negativt heltal j. I beviset for sætning 11 under "hvis", husker vi, at

$$n = n_1 \cdot n_2 = \left(\frac{n_1 + n_2}{2}\right)^2 - \left(\frac{n_1 - n_2}{2}\right)^2 \Rightarrow \left(\frac{n_1 - n_2}{2}\right)^2 = \left(\frac{n_1 + n_2}{2}\right)^2 - n.$$

Hvis man har $n = n \cdot 1$, får vi (sæt $n_1 = n$ og $n_2 = 1$)

$$\left(\frac{n-1}{2}\right)^2 = \left(\frac{n+1}{2}\right)^2 - n$$

og der vil være et kvadrattal i følgen, med $(k+j)^2 - n = ((n+1)/2)^2 - n$, dvs. $k + j = (n + 1)/2$. Heraf viser det, at n er et primtal, idet n har den trivielle faktorisering $n = n \cdot 1$.

Eksempel: Lad $n = 31861$. Hvis x rundes ned til næste hele tal, skriver vi $\lfloor x \rfloor$. Hvis x rundes op til næste hele tal, skriver vi $\lceil x \rceil$. Vi ser, at

$$\left\lfloor \sqrt{31861} \right\rfloor^2 < 31861 < \left\lceil \sqrt{31861} \right\rceil^2 \Leftrightarrow 178^2 < 31861 < 179^2$$

Vi ønsker nu, at finde et kvadrattal, ved at indsætte $k = 179$ i følgen. Der findes et kvadrattal ved tredje skridt, idet:

$$179^2 - 31861 = 180$$

$$(179 + 1)^2 - 31861 = 539$$

$$(179 + 2)^2 - 31861 = 900 = 30^2$$

Derved fås:

$$31861 = ((179 + 2) + 30) \cdot ((179 + 2) - 30) = 211 \cdot 151$$

De 2 primfaktorer for tallet, er x = 211, og y = 151. Bemærk, at primfaktorerne, 211 og 151 er næsten lige store, hvorfor der kun skulle bruges 3 skridt til at faktorisere tallet, men hvad hvis primfaktorerne ikke er næsten lige store:

Eksempel: I dette eksempel bruges primtallene, $x = 17$ og $y = 103$. Produktet af disse to primtal giver tallet 1751 hvilket er ca. $18{,}20$ gange mindre end det tal vi faktoriserede før, 31861. Dermed bør faktorisering af tallet 1751 kræve færre udregninger end før:

Kvadratroden tages af tallet:

$$\left\lfloor \sqrt{1751} \right\rfloor^2 < 1751 < \left\lceil \sqrt{1751} \right\rceil^2 \Leftrightarrow 41^2 < 1751 < 42^2.$$

Indsættes k = 42 i følgen, får vi:

$$\{42^2 - 1751, 43^2 - 1751, 44^2 - 1751, \dots, (42 + j)^2 - 1751, \dots\}$$

for alle ikke-negative heltal j. Efter 19 skridt, har man et kvadrattal, idet $60^2 - 1751 = 1849 = 43^2$:

$$1849 = (60 + 43) \cdot (60 - 43) = 103 \cdot 17$$

Fermats metode virker bedst, hvis de to primtal er tæt på hinanden i størrelsen. Det vil sige, at hvis n_1 og n_2 er næsten lige store, så bliver y lille. Derfor skal der foretages færre udregninger, jo tættere på, de to primtal er i størrelsen.

Primtalsfaktoring[14]

Sætning 12: For alle positiv heltal n > 1 kan skrives som et produkt af primtal på en unik måde.

Bevis: *Eksistensdelen;* Vi beviser det ved modstrid. Antag, at der findes et n > 1, der ikke kan skrives som et produkt af primtal. Hvis det er tilfældet, så ville alligevel n være faktoriseret i et produkt af primtal. Der findes derfor et divisor q i n, hvor q er forskelligt fra 1 og n, således at $n = qa$.

Både a og q har faktoriseringer i produkter af primtal, da $1 < d < n$ og $1 < q < n$ mens n er antaget ikke at have primtalsfaktorisering. Ifølge denne ligning, $n = qa$, følger jo, at n rent faktisk kan primtalfaktoriseres ved at samle faktoriseringer af q og d på højre side. Det vil sige, at der ikke findes et $n > 1$, der ikke kan skrives som et produkt af primtal.

Entydighedsdelen; Lad $p_1p_2p_3 \dots p_i = q_1q_2q_3 \dots q_i$ være to primtalsfaktoriseringer af n. Vi ser på venstre side. Da primtallet $p_1$er divisor i $p_1p_2p_3 \dots p_i$, er det også divisor i $q_1q_2q_3 \dots q_i$, så må $p_1$være divisor i en af primfaktorerne på $q_1q_2q_3 \dots q_i$. Eftersom primtallet, som en divisor, kun har 1 må p_1 være lig med primfaktorerne i $q_1q_2q_3 \dots q_i$. Forkortes denne fællesfaktor væk, gentager vi ved at gøre det på samme måde med $p_2p_3 \dots p_i$ som før. Konklusionen er, at primfaktorerne på venstreside optræder i højreside mindst det samme antal gange. Hvis vi ser på højre side, og gentager den samme argumentation, får vi konklusionen, at primfaktorerne på højre side optræder i venstre mindst det samme antal gange. Samler man de to konklusioner sammen, har vi hermed vist, at en primfaktor indgår et entydigt antal gange i faktoriseringen.

Primtalsfaktorings tidsforbrug[15]

For at få en idé om, hvor tidskrævende det kan være, at primfaktorisere et stort tal. Så antager vi, at vi ønsker, at primfaktorisere en 1024-bit nøgle dvs. et tal med ~309 cifre. Hvis vi antager, at det

[14] Hansen, Johan, m.fl. Algebra og talteori, s. 42
[15] Hansen (14. nov 2011) http://goo.gl/hAiYVG "Deling - primtal - kryptografi" side 5, Lokaliseret 11/12/2014

tages 1 millionte del af et sekund, om at afgøre, om et helt tal er i divisor i et andet tal, og dette skal gøres 10^{309} gange så vil det tage:

$$10^{309} \cdot 10^{-6}\ sekund = 10^{303} \cdot \frac{1}{60 \cdot 60 \cdot 24 \cdot 365}\ år \sim 3{,}2 \cdot 10^{295}\ år$$

Her bliver RSA-sikkerhed belyst, hvor det tager langtid, om at primfaktorisere store tal. Dette kigger vi på længere fremme.

Program med Fermats metode

Jeg har lavet et program, der bruger fermats metode til at faktorisere et tal. Programmet giver som output, primfaktorene af tallet og antal sekunder der skal bruges for at faktorisere tallet.

Eksempel:

Tallet 2096609 faktorieres vha. programmet:

```
class fermat {

    public static function factorize

    $time = microtime();
    $time = explode(' ', $time);
    $time = $time[1] + $time[0];
    $start = $time;
    $steps = 0;

    $a = ceil(sqrt($number));
    $b2 = $a*$a - $number;
    $fermat = array();
```

Figur 2 - et udpluk af koden. Programmet er lavet i programmeringssproget PHP. Se bilag 4 for hele koden.

```
$numberToFactorise = 2096609;
$factorise = $fermat->factorize($numberToFactorise);
print_r($factorise);
```

Figur 3 - Programmet køres ved at kalde på funktionen factorize med tallet n, som er det tal der skal faktoriseres.

Output:

```
Array ( [x] => 1117 [y] => 1877  => [timeInSecounds] => 0.0001 )
```

Figur 4 - Programmets output af faktorisering af tallet 2096609. Faktorisering udført d. 09/12/2014.

Det tog 0.0001 sekunder om at faktorisere tallet, hvor de to primfaktorer er x = 1117 og y = 1877. Programmet vil blive benyttet til at analysere sammenhængen mellem antal bits og den tid der skal bruges for at faktorisere tallet.

RSA-Kryptering[16]

I dette afsnit konstrueres en hemmelig- og offentlig nøgle. Der vil blive vist et eksempel, hvor ordet "**SRP**" krypteres, hvorefter den hemmelige nøgle bruges til at dekryptere chifferteksten.

RSA-systemet har en nøgleberegning, der består af følgende skridt:

1) Sæt n = pq, hvor to primtal p og q vælges.
2) Beregn $\varphi(n)$.
3) Vælg et heltal e som opfylder $0 < e < \varphi(n)$, og at e og $\varphi(n)$ er indbyrdes primiske.
4) Beregn d, således $e \cdot d \equiv 1 \, mod \, \varphi(\mathrm{m})$

Vi har nøglerne, den offentlige (n, e) og den hemmelige d.

Enkryptering med RSA: Tallet $m \in]0, n[$ enkrypteres ved $m \longrightarrow m^e \pmod n = c$.

Dekryptering med RSA: Tallet c dekrypteres ved $c \longrightarrow c^d \pmod n = m$.

Der bruges følgende tabel, til at lave bogstaverne om til tal:

A	B	C	D	E	F	G	H	I
01	**02**	**03**	**03**	**04**	**05**	**06**	**07**	**08**
J	K	L	M	N	O	P	Q	R
09	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**
S	T	U	V	W	X	Y	Z	
18	**19**	**20**	**21**	**22**	**23**	**24**	**25**	

[16] Landrock, Peter, m.fl. *Kryptologi – fra viden til videnskab*. Side 102-105

Vi kunne sagtens have ladet talrækken være anderledes, fx A = 10, B = 22, C = 05, men for overskueligheden skyld bruges A = 01, B = 02 […].

Ordet omskrives til tal: S = 18, R = 17, og P = 15.

Enkryptering

1) To primtal vælges: $p = 53$ og $q = 47$ Produktet af primtallene giver: $m = 53 \cdot 47 = 2491$ (vores offentlige nøgle, denne må gerne deles på Facebook)
2) Vi har $\varphi(\mathrm{m}) = (53 - 1) \cdot (47 - 1) = 2392$.
3) Da $(2491{,}25) = 1$, er tallene er indbyrdes primiske. Udregninger findes i bilag 1.
4) $d = 2105$ (vores hemmelige nøgle) Udregninger findes i bilag 1.

Vi ser på $m \longrightarrow m^e \ (mod\ n)$, dvs. $m \longrightarrow m^{25} \ (mod\ 2491)$

Sætter vi m = 18, 17 og 15, har vi ved brug af Wolfram Alpha:

S = mod(18^{25}, 2491) = 1640

R = mod(17^{25}, 2491) = 2357

P = mod(15^{25}, 2491) = 2219

Vi kunne selv have regnet det ud vha. talteorien, men i praksis benytter man sig af IT-systemer, der gør dette automatisk, hvorfor der i dette tilfælde er brugt Wolfram Alpha.

Dekryptering

$c \longrightarrow c^d \ (mod\ n)$ dvs. $c \longrightarrow c^{2105} \ (mod\ 2491)$

S = mod(1640^{2105}, 2491) = 18

R = mod(2357^{2105}, 2491) = 17

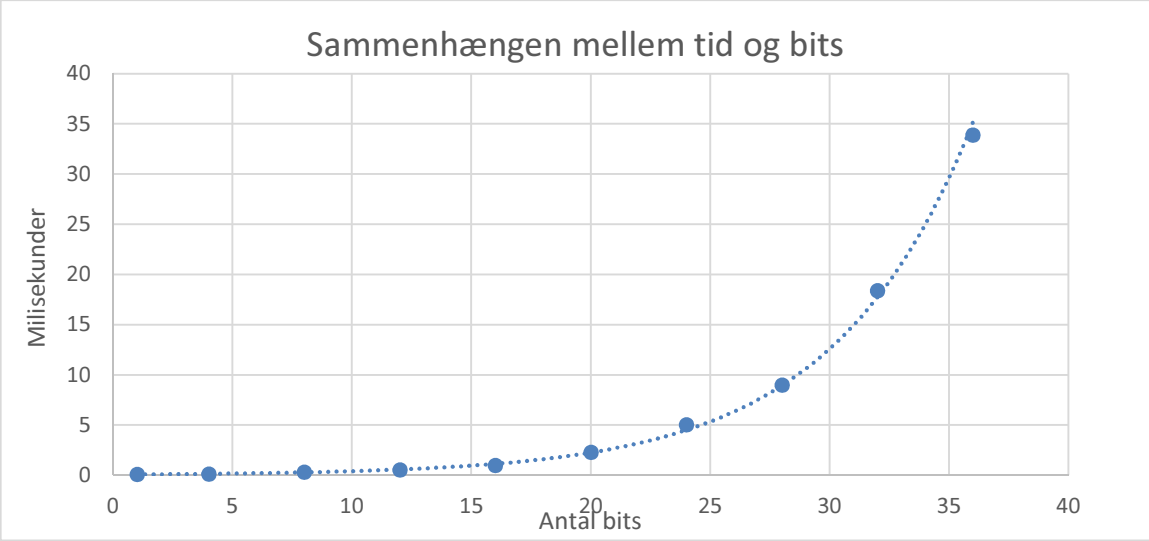
P = mod(2219^{2105}, 2491) = 15

Tallene omskrives til bogstaver, hvorved vi får ordet **"SRP"**

Bits og tid

Der er blevet vist, at det tager langtid at primfaktorisere store tal. For at komme dybere ind på området, så har jeg lavet en analyse, der skal se sammenhængen mellem bits og den tid der skal bruges for faktorisere et tal. Til analysen, har jeg benyttet mig af mit program med Fermats metode.

Fermats metode virker bedst, hvis de to primtal næsten er lige store. Derfor er der til denne analyse brugt to primtal, hvor det ene primtal er 2 gange mindre i størrelsen end det andet primtal. Det antages, at man i praksis ved, at der findes en algoritme, der er i stand til at finde primfaktorerne *hurtigt*, hvis de to primfaktorer har næsten samme størrelse, hvorfor man i praksis ikke benytter sig af to primfaktorer der er næsten lige store:



Figur 5 - Analysen er kørt på en computer med en Intel Core i7-4930MX processor, 3 GHz med 16 GB ram, d. 10/12/2014. På x-aksen har vi antal bits og på y-aksen har vi tiden i millisekunder. Analysen er kørt for hver 4. bit op til 36 bits.

Der ses på grafen, at der er tale om en eksponentiel udvikling. Der ses, at bit-nøglens størrelse påvirker den tid det tager, at faktorisere et tal. Jo større bit-nøglen er, jo længere tid tager faktoriseringen af tallet. Dermed giver denne sammenhæng en klargøring af RSA-kryptosystemets sikkerhed, hvor bit-nøglens størrelse gør metoden mere sikker, jo større nøglen er. Jeg har vha. regression opstillet en eksponentiel funktion for grafen:

$$f(x) = 0{,}07444 \cdot 1{,}1867^{x}$$

Fremskrivningsfaktoren for funktionen er 1,867 dermed er vækstprocenten 18,67%. Det betyder, at det tager 18,67% længere tid om at faktorisere tallet for hver bit den vokser med. X-værdien i funktionen er antal bits. I praksis benytter man sig af 1024 bit-nøgler eller større. Hvis jeg skulle faktorisere et tal på 1024 bit vha. fermats metode, så ville det tage mig:

$$\sim 3{,}1 \cdot 10^{65} \text{ år - (se bilag 6 for udregninger)}$$

til sammenligning, så brænder solen ud om 7,59 mia. år[17]

Dermed ses der, at der er en sammenhæng mellem bits og den mængde tid der skal bruges, for at faktorisere et tal. Derfor er det med god grund, at det endnu ikke er lykkedes nogen, at faktorisere en 1024 bit-nøgle, selvom der er en præmie på 100,000$, hvis det skulle lykkes nogen.[18] Den største kendte bit-nøgle der er blevet faktoriseret, er en bit-nøgle på 768-bits, hvor det lykkedes et internationalt hold bestående af matematikere, dataloger og kryptologer at faktorisere en 768-bit nøgle d. 12 december 2009. Benjamin Jun som var en del af holdet, sagde følgende om forløbet: ”*The team managed to factor the 232-digit number that RSA held out as a representative 768-bit modulus from a now-obsolete challenge. They spent half a year using 80 processors on polynomial selection. Sieving took almost two years and was done on "many hundreds of machines". Using a single-core 2.2GHz AMD Opteron with 2GB RAM, sieving would have taken about 1,500 years, they estimated.*”[19]

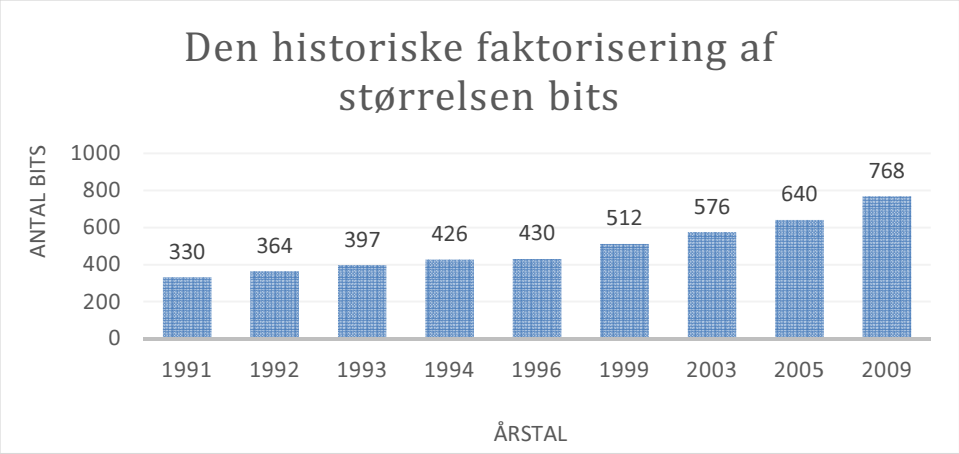
Med denne udtalelse viser det, at det ikke er alle, som kan faktorisere et stort tal[20]. Selvom holdet bestod af en masse dygtige folk, så tog det dem sammenlagt 2,5 år om at faktorisere en 768-bit nøgle. Hvis det skulle gøres på en ”almindelig computer”, så ville det tage 1500 år eller $\sim 3.1 \cdot 10^{65}$ år, alt efter hvilken algoritme der benyttes. Ifølge Benjamin Jun, så er det 1000 gange sværere at faktorisere en 1024-bits nøgle i forhold til en 768-bit nøgle, men det betyder ikke, at en 1024-bit nøgle er sikker i fremtiden. Det blev anbefalet, at man i 2010 skulle gå fra 1024-bits nøgler til 2048-bits nøgler. Fordi det lykkedes holdet året inden, at faktorisere en 768-bit nøgle. Anbefalingen lyder på, at en 2048-bit nøgle bruges til året 2030[21].

[17] Egekvist (2008) http://goo.gl/kxos9J “Jorden går under - Om 7,59 milliarder år” Lokaliseret 12/12/2014
[18] Wikipedia (2014) http://goo.gl/jBh8yf "RSA Factoring Challenge” Lokaliseret 12/12/2014
[19] Goodin (2009) http://goo.gl/l81Plz "RSA crypto defiled again, with factoring of 768-bit keys” Lokaliseret 12/12/2014
[20] Simpsons (1997) http://goo.gl/l81Plz "Key Length - How Long is Long Enough?" Lokaliseret 12/12/2014
[21] Coffey (2012) http://goo.gl/5V2U1y "RSA key lengths" Lokaliseret 13/12/2014

Nøglerne bliver større over tid

Størrelsen af bit-nøglerne, der bruges til RSA-kryptering er vokset gennem tiden. Der er kommet nye algoritmer, som langt hurtigere kan faktorisere et tal i forhold til fermats metode fx Pollards ρ-algoritme[22]. Samt er processorkraften også afgørende for, hvor hurtig algoritmen er til at faktorisere et tal. Netop som Benjamin Jun udtalte, at hvis de havde forsøgt sig med at faktorisere en 768-bit nøgle, med en processor med 1 kerne med 2.2GHz, så ville det have taget ca. 1500 år om at faktorisere tallet.

Følgende figur viser, at siden 1991 er det lykkedes, at faktorisere større bits-nøgler gennem årene[23].



Figur 6 - Gennem årene er det lykkes at faktorisere større bit-nøgler.

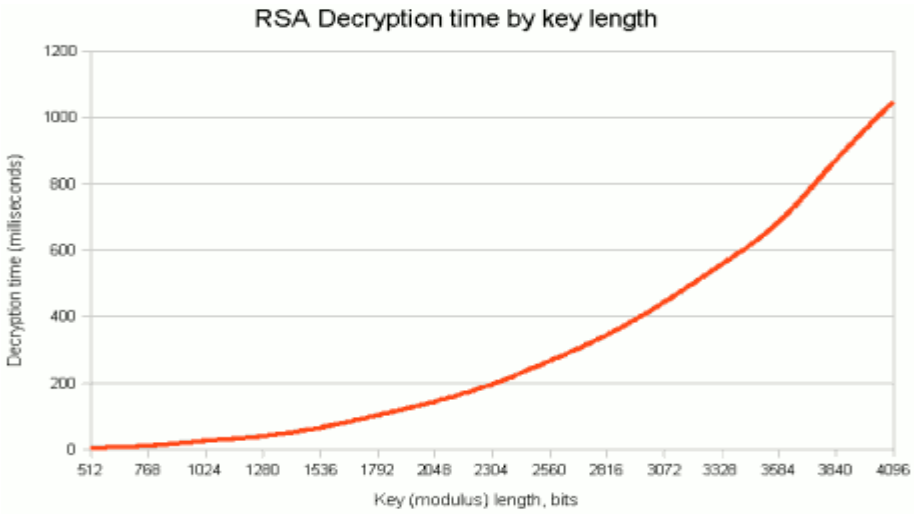
Figuren belyser også, at størrelsen af de nøgler, som bruges til RSA vokser. Fordi hvis nogen har faktoriseret et x-antal bits, så er det en sikkerhedsrisiko at benytte en talstørrelse som før er faktoriseret. Derfor bruges der en talstørrelse, som er større end det kendte talstørrelse der er blevet faktoriseret. Dette giver god mening, når sikkerhedseksperter anbefaler, at man i dag benytter en bit-størrelse på 1024 eller 2048 bits fordi disse endnu ikke er blevet faktoriseret. Der bliver ikke anbefalet, at bruge en talstørrelse 768 bits eller mindre, fordi disse før er blevet brudt.[24] Figuren belyser også, at det er et spørgsmål om tid, før det er muligt, at faktorisere en talstørrelse, som før ikke er faktoriseret. Eksempelvis, havde man ikke faktoriseret en 768-bits nøgle før året 2009. Dermed er talstørrelsen der

[22] Barnes (7 dec 2004) http://goo.gl/hBNjEV "Integer Factorization Algorithms" side 12, Lokaliseret 13/12/2014
[23] Wikipedia (2014) http://goo.gl/UapDxA "RSA Factoring Challenge" Lokaliseret 13/12/2014
[24] EMC (?) http://goo.gl/OlNmIA " WHAT KEY SIZE SHOULD BE USED" Lokaliseret 13/12/2014

benyttes praksis vokset gennem tiden, hvor størrelsen fortsat vokser over tid, for at sikkerheden bibeholdes i fx banker ellers vil det være en sikkerhedsbrist, hvis der bliver benyttet bit-nøgler i IT-systemer, som før er blevet brudt[25]

Hvorfor gør man ikke nøglerne 'kæmpe' store?[26]

Hvorfor benytter man sig ikke bare af kæmpe store bit-nøgler, således at IT-systemer bliver mere sikre? Ved at bruge store bit-nøgler, så bliver dekrypteringsprocessen langsommere, det skyldes at chifferteksten bliver længere når bit-nøglen vokser.



Figur 7 - Figuren viser, at dekrypteringsprocessen bliver langsommere, når bit-størrelsen vokser. Analysen er foregået på en 2GHZ Pentium processor. Kilde: fodnote 25.

Ved hver fordobling af bit-størrelsen, bliver dekrypteringsprocessen 6-7 gange langsommere. Krypteringsprocessen bliver også langsommere, men det er man ikke så meget interesseret i. Fordi hele dekrypteringsprocessen foregår på serveren, eksempelvis, når man logger på netbank. Hvis banken benytter sig af en 4096 bit-nøgle, så tager det ca. 1 sekund af CPU'ens tid for at dekryptere chifferteksten, ved hver login, det lyder ikke af meget, men hvis en hacker begynder med at sende tilfældige data til netbankens login-system, så bliver der ved hver login-forsøg 'brændt' ét sekund af CPU'ens levetid og dermed kan en hacker 'ødelægge' serveren vha. af metoden. Med en 1024-bit nøgle tager dekrypteringsprocessen ca. 25 millisekunder, og ved at benytte login-restriktioner på antal login-forsøg, der tillades pr. bruger gør, at det bliver nemmere at beskytte CPU'en imod 'bruteforce-angreb'.

[25] Coffey (2012) http://goo.gl/5V2U1y "RSA key lengths" Lokaliseret 13/12/2014
[26] Fodnote 26.

Dermed betyder det, at CPU'ernes hastighed, der er tilknyttet serveren spiller en rolle, når der skal overvejes en forøgelse af bit-nøglens størrelse i IT-systemer.

Fremtiden med RSA

For at kunne sige noget om fremtiden, så lad os først se mod nutiden. RSA-kryptosystemet er fortsat sikkert, hvis nøglerne er tilpas store. Metoden er dog til gengæld kompliceret og krævende ved store mængder data, fordi den generede chiffertekst, bliver mange gange længere end det oprindelige data, hvilket betyder, at dekrypteringsprocessen bliver langsommere. Det betyder ikke, at man ikke kan tillade sig at 'skrue op' for nøglernes størrelse. CPU'erne vil i fremtiden blive hurtigere, hvorfor de vil være i stand til at kunne følge med nøglernes størrelsesorden.[27]

Det ser heller ikke ud til, at RSA-kryptosystemet bliver svækket over tid, fordi bedre processorer åbner op for hurtigere computer, hvor det bliver muligt, at generere større nøgler[28]. Det største kendte primtal til dato er på 17425170 cifre[29] til sammenligning består en 1024-bit nøgle på 308 cifre. Dermed ser det ud til, at den eneste måde hvorpå RSA kan svækkes på, er opfindelse af kvantecomputeren, der kan faktorisere et tal i polynomiel tid.

Det er derfor svært, at afgøre om det overhovedet vil lykkes nogen, at finde en nem og praktisk metode til at bryde RSA, hvis det skulle ske, så beskriver følgende citat, hvilke konsekvenser det kan få for verdenssamfundet: "*If an easy and practical solution were found, the RSA code would be broken and the prize is, well, most of the world's bank accounts.*[30]" Det er ikke kun banker, som vil kunne blive udsat for angreb - det er stort set alle systemer hvor RSA er implanteret i sikkerhedssystemerne. Det er svært, at forestille sig de præcise konsekvenser det kan få for verdenssamfundet, hvis en metode skulle finde sted. Mit bud på dette vil være, at konsekvenerne vil blive katastrofale. Det bliver muligt for potentielle hackere, at til skaffe sig adgang til folkets bankkontoer og andre fortrolige oplysninger, hvor disse kan udnyttes. Dermed bliver folkets tillid til samfundet ikke længere-eksisterende, fordi alt det vi troede som før var fortroligt, vil nu være frit tilgængeligt for alle. Derfor skal der bygges en ny digital samfundsstruktur op, hvorved de enkelte landes politikere skal bruge en masse kræfter på, at genvinde borgernes tillid - spørgsmålet er, om det bliver muligt?

[27] Anthony (2012) http://goo.gl/LdEtJK "CPUs of the future: AMD partners with ARM, Intel designs a brain on a chip" Lokaliseret 13/12/14
[28] Curting (2011) http://curting.com/fremtiden.php "Fremtiden" Lokaliseret 14/12/14
[29] Wikipedia (2014 dec) http://goo.gl/cM7oH1 "Largest known prime number" Lokaliseret 14/12/14
[30] Kielpinski (11 feb 2014) http://goo.gl/JszLyn "Quantum computers could crack [..]" Lokaliseret 14/12/14

Konklusion

RSA er et asymmetrisk krypteringssystem, hvor der er to sæt nøgler, en offentlig og en privat nøgle. Den offentlige nøgle bruges til at kryptere en tekst, mens den private nøgle bruges til at dekryptere chifferteksten. Den private nøgle kendes kun af ejeren, mens alle har adgang til den offentlige nøgle. Metodens sikkerhed bygger på, at det er svært tidsmæssigt, at primfaktorisere store tal. Problemet bliver sværere, hvor det tager længere tid, at primfaktorisere et tal, jo større tallet det er. Der findes uendelige antal primtal, hvorfor nøglernes størrelse kan øges i takt med at processerne bliver hurtigere.

De nøgler, som bliver brugt i RSA-kryptering er i løbet af årene vokset, og vil fortsat vokse i løbet af tiden. Det skyldes, at processer kraften bliver hurtigere i løbet af årene, hvilket gør, at faktoriserings algoritmer bliver hurtigere, hvorfor det i løbet af årene bliver muligt, at faktorisere større bit-nøgler end hvad der tidligere har været muligt. Derfor skal bit-nøglernes størrelse øges i IT-systemer i løbet af årene, ellers er det en sikkerhedsrisiko, at benytte nøgler, som før er blevet brudt.

Pierre de Fermat's faktoriserings algoritme er effektiv, hvis tallet har to primfaktorer, der er næsten i samme størrelse. Metoden er upraktisk, ude i "den virkelige verden", da metoden er langsom ved faktorisering af store tal. Der er nyere faktoriserings algoritmer, som er mere effektive end Fermats metode, men end disse løser ikke faktoriseringsproblemet. Det er uvist, om det vil lykkes nogen, at skabe en metode. Det kan få alvorlige konsekvenser for verdenssamfundet, hvor potentielle hackere kan få adgang til b.la. borgernes- bank kontoer og fortrolige oplysninger, hvis en metode skulle finde sted.

Litteraturliste

Landrock, Peter og Knud Nissen: Kryptologi - fra viden til videnskab. Anvendte sider 71-73, 87-88, 94-95, 129-130, 102-105. 2. udg. ABACUS, 2014. (eBog)

WHAT KEY SIZE SHOULD BE USED Udgivet af RSA LABORATORIES. Internetadresse: http://goo.gl/OlNmIA - Besøgt d. 13.12.14 (Internet)

RSA Factoring Challenge. Udgivet af Wikipedia. Internetadresse: http://goo.gl/BZ5tXZ - Besøgt d. 13.12.214 (Internet)

Integer Factorization Algorithms. Udgivet af Barnes. Internetadresse: http://goo.gl/hBNjEV - Besøgt d. 13.12.14 (Internet)

EGEKVIST, KIRSTIN: *Jorden går under - Om 7,59 milliarder år*. I: Avis, 2008, s. 1. Internetadresse: http://goo.gl/vNPTUS Besøgt d. 12.12.14 (Artikel)

Key Length - How Long is Long Enough. Udgivet af Sarah Simpsons. Internetadresse: http://goo.gl/l81Plz - Besøgt d. 12.12.14 (Internet)

RSA key lengths. Udgivet af Neil Coffey. Internetadresse: http://goo.gl/5V2U1y - Besøgt d. 13.12.14 (Internet)

Deling - primtal - kryptografi. Udgivet af Johan P. Hansen. Internetadresse: http://goo.gl/hAiYVG - Besøgt d. 11.12.2014 (Internet)

Bevisteknikker. Udgivet af RUC. Internetadresse: http://goo.gl/h8sli5 - Besøgt d. 11.12.14 (Internet)

HANSEN, JOHAN og HENRIK SPALK: ALGEBRA OG TALTEORI. Anvendte sider, 42-43 og 15-16 1. udg. Nordisk forlag, A.S., Copenhagen, 2002. (Bog)

GREATEST COMMON DIVISOR. Udgivet af Courant Institute of Mathematical Sciences . Internetadresse: http://goo.gl/ARpPcz - Besøgt d. 9.12.14 (Internet)

RSA-kryptosystemet. Udgivet af Erik Vestergaard. Internetadresse: http://goo.gl/HF28DX - Besøgt d. 7.12.2014 (Internet)

Sikkert og pålideligt peer-to-peer filsystem. Udgivet af Jacob Nittegaard-Nielsen . Internetadresse: http://goo.gl/sJvkB4 - Besøgt d. 04.12.14 (Internet)
Damhus: Informationsteknologi. 1. udg. Systime A/S, 2013. (Bog)

Largest known prime number. Udgivet af Wikipedia. Sidst opdateret: 13.12.2014. Internetadresse: http://goo.gl/3sH9N - Besøgt d. 12.12.14 (Internet)

Anthony, Sebastian: *CPUs of the future: AMD partners with ARM, Intel designs a brain on a chip*. I: Avis, 2014, s. 1-1. Internetadresse: http://extremetech.com/computing/131230-cpus-of-the-future-amd-partners-with-arm-while-intel-designs-a-brain-on-a-chip Besøgt d. 12.12.14 (Artikel)

Kielpinski, David: *Quantum computers could crack existing codes but create others much harder to break*. I: http://theconversation.com/, 11.02.2014. Internetadresse: http://goo.gl/JszLyn Besøgt d. 12.12.14 (Artikel)

Fremtiden. Udgivet af Oliver Curting. Sidst opdateret: 2011. Internetadresse: http://curting.com/fremtiden.php - Besøgt d. 11.12.14 (Internet)

RSA key lengths. Udgivet af Neil Coffey. Internetadresse: http://goo.gl/qdqEz2 - Besøgt d. 12.12.14 (Internet)

Bilag

Bilag 1

Største fælles divsior (enkryptering):

$$2392 = (95 \cdot 25) + 17$$

$$25 = (1 \cdot 17) + 8$$

$$17 = (2 \cdot 8) + 1$$

(2392, 25) = 1 dvs. indbyrdes primiske. Vi kunne også have brugt, mit program fra bilag 6, til at finde et indbyrdes primisk tal.

Euklids udvidede algoritme[31]

Euklids udvidede algoritme, kan bruges til at skrive en linearkombination af a og b som den største fælles division. Der findes hele tal x og y, så:

$$(a, b) = (a \cdot \mathrm{x}) + (b \cdot \mathrm{y})$$

Denne algoritme kaldes for Euklids udvidede algoritme. Tallene x og y er ikke entydige bestemte dvs. der findes forskellige x og y værdier, som løser problemet. Med Euklids udvidede algoritme, kan vi bestemme et talsæt af x og y der passer. Det skal nævnes, at tallene x og y ikke behøves at være positive.

Beregne den hemmelige nøgle:

Vi skal finde "u", hvor vi går baglæns i vores mellemregninger, fra euklids algoritme dette gøres vha. euklids udvidede algoritme.[32]

Vi skal finde den inverse til 25 $modulo$ 2392: (vi bruger vores viden fra talteorien, inverse elementer)

$$25 \cdot \mathrm{d} \equiv 1 \, mod \, 2392$$

Vi benytter os af euklids udvidede algoritme, hvor nedenstående ligning skrives med en konstant, u:

[31] Hansen, Johan, m.fl. Algebra og talteori, s. 15-16
[32] http://curting.com/RSA-teori.php Afsnittet dekryptering. Lokaliseret 14/12/14

$$d \cdot 25 = u \cdot 2392 + 1 \Leftrightarrow 25 \cdot d + (-u) * 2392$$

Vi bruger mellemregningerne fra bilag 1 (euklids algoritme) til at beregne den hemmelige nøgle, d:

$17 = 1 \cdot 2392 + (-95) \cdot 25$

hvorved denne ligning gør en anden ligning til:

$$8 = 25 - 1 \cdot (1 \cdot 2392 + (-95) \cdot 25) = (-1) \cdot 2392 + 96 \cdot 25$$

Den anden sidste ligning gøres til:

$$1 = 1 \cdot (1 \cdot 2392 + (-95) \cdot 25) + (-2) \cdot \big((-1) \cdot 2392 + 96 \cdot 25\big) = 3 \cdot 2392 + (-287) \cdot 25$$

En linearkombination af $y = -287$ og $x = 3$ haves for $(25, 2392)$:

$$(-287) \cdot 25 \equiv 1 \bmod 2392$$

-287 er den inverse til 25 $modulo$ 2392. Da den inverse har et negativt fortegn, gøres der et par ekstra beregninger:

$$(n - d) \cdot e \equiv 1 \bmod n$$

$$(2392 - 287) \cdot 25 \equiv 1 \bmod 2392 \Leftrightarrow 25 \cdot 2105 \equiv 1 \bmod 2392$$

Den hemmelig nøgle er, $d = 2105$

Et godt værktøj til at beregne den hemmelige nøgle kan findes på følgende link:

http://asecuritysite.com/Encryption/rsa

Bilag 3

Bestemmelse af $3^7 \bmod 5$

Først bestemmer vi:

$3^1 \equiv 3 \bmod 5$

$3^2 = (3^1)^2 = 9 \bmod 5$

$3^4 = (3^2)^2 = 4^2 = 16 = 1 \bmod 5$

Opskriver vi 7 = 4 + 2 + 1 får vi vha. potensreglerne:

$3^7 = 3^4$

Bilag 4

Programmet kan køres på følgende side: Erstat x, med det tal, som skal faktoriseres.

http://89.239.247.100/other/folders/algoritms/cryptography/RSA/fermat.php?n=x&access=SRP&phpversion=4.4

Fermats metode er udviklet i programmeringssproget php:

```
1. class fermat { // opretter class
2.
3.         public static function factorize($number) { // opretter funktion, med variablen $number, som er det tal, der skal faktoriseres.
4.
5.         //begynder at tælle tid
6.         $time = microtime();
7.         $time = explode(' ', $time);
8.         $time = $time[1] + $time[0];
9.         $start = $time;
10.
11.
12.             $x = ceil(sqrt($number)); // kvadratroden af tallet $number tages (sqrt), hvorefter tallet rundes op (ceil)
13.             $y2 = $x*$x - $number; // variablen $x opløftes 2 gange, og bliver minusset med $number (tallet som skal faktoriseres). Ligesom i eksemplet.
14.             $fermat = array(); // opretter et array, der gemmer al output.
15.
16.                     while(gmp_perfect_square($y2) == false) { // Bruger PHP-funktion perfet square der tjekker om $y2 er et kvadrattal. Hvis ikke, så fortsætter løkken indtil tallet er et kvadrattal.
17.
18.                             $x += 1; // vi benytter os af følgen, der lægges +1 til $x.
19.                             $y2 = $x*$x - $number; // $y2 opdateres, med de nye data. Næste gang løkken kæres, tjekkes der om $y2 er et kvadrattal. Akkurat som følgen.
20.
21.                     }
22.
23.                     // tallet $number er faktoriseret, hvorfor vi stopper med at tælle hvor langtid det tog om at faktorisere tallet.
24.                     $time = microtime();
25.                     $time = explode(' ', $time);
26.                     $time = $time[1] + $time[0];
27.                     $finish = $time;
28.                     $total_time = round(($finish - $start), 4);
29.
30.                     //gemmer outputs i arrayet, [x, y, og tid i sekunder - i analysen har jeg konventeret sekunder til [ms].
31.                     $fermat['x'] = $x - sqrt($y2);
```

```
32.                     $fermat['y'] = $x + sqrt($y2);
33.                     $fermat['timeInSecounds'] = $total_time;
34.
35.                     return $fermat; // outputter arrayet $fermat.
36.
37.             } // slutter funktion.
38.
39.     } // slutter class,
40.     $numberToFactorise = 0; // tallet som skal faktoriseres.
41.     $fermat = new fermat(); // intialisere fermat class
42.     $factorise = $fermat->factorize($numberToFactorise); // kontakter
    funktionen factorize med tallet der skal faktoriseres ($numberToFactorise)
43.     print_r($factorise); // outputter $fermat array
```

Bilag 5

Euklids algoritme til at beregne den største fælles divisor: Udviklet i programmeringssproget php.

```
1. class euklid { // class oprettes.
2.
3.         public static function gcd($a, $b) { // funktionen gcd(a,b)
    oprettes.
4.
5.         $return = $a; // denne variabel bruges til at returnere den stør-
    ste fælles division.
6.
7.         $return_a = $a; // gemmer $a, bruges til at udskrive $a værdien
    senere hen.
8.         $return_b = $b; // gemmer $b, bruges til at udskrive $b værdien
    senere hen.
9.
10.             while($b != 0) { // løkken køres indtil resten er lig med 0.
11.
12.                     $t = $b; // variablen $t får værdien $b
13.                     $b = $return % $b; // modulos af tallet $b tages
    (bruges til når løkken køres igen)
14.                     $return = $t; // returnerer største fælles divisior
15.
16.             }
17.
18.             if($return == 1) { // Den største fælles divison er = 1, der-
    for er $a og $b indbyrdes primiske.
19.
20.                     $return = $return.' dvs. Indbyrdes primisk'; //
21.
22.             } // slutter if sætning.
23.
24.                     return 'Største fælles divisor gcd('. $return_a.',
    '.$return_b.') = '.$return; // returnerer største fælles visior. for
    gcd(a,b)
25.
26.             } // slutter funktionen.
27.
28.     } // slutter ”class”
29.
30.     $euklid = new euklid(); // initialiserer class "euklid"
```

```
31.     $euklid = $euklid->gcd(23,29700); // funktionen gcd køres, med a = 23
   og b = 29700.
32.     echo $euklid; // outputter $return fra variablen $euklid
```

Output:

Største fælles divisor gcd(23, 29700) = 1 dvs. Indbyrdes primisk

Bilag 6

Disse udregninger er regnet i det matematiske program, TI interactive.

$f(x) := 0.7444 \cdot 1.1867^{x}$ (milli sek)

Antal sek for at primfaktoriserer en X bit nøgle.

$$\text{convertToSec}(bits) := \frac{f(bits)}{1000}$$

Antal år for at primfaktoriserer en 1024 bit nøgle.

$$\frac{\text{convertToSec}(1024)}{(24 \cdot 60 \cdot 60) \cdot 365} \quad <=> 3.14865e65$$

3.1 * 10^65 år.