Sikkerhed bag login-formular på en hjemmeside

Jens Tinggaard

4. december 2019

Resumé

her står hvad det kommer til at handle om...

Indholdsfortegnelse

Introduktion / Forord

Motivation samt formalia – kryptologi og kryptografi er det samme!

RSA

Uden nødvendigvis at være klar over det, er vi alle afhængige af RSA, det er en helt fundamental ting af vores hverdag. RSA er grunden til at man kan handle på nettet, uden at få stjålet sine kreditkortoplysninger. RSA-krytpografi ligger også til grunde for, at man ikke blot kan lave online identitetstyveri.[Ves07]

RSA er en metode, brugt til at kryptere data, døbt efter sine tre stiftere: *Ron Rivers*, *Adi Shamir* og *Len Adleman*, tilbage i 1977.[Ves07] Metoden er baseret på antagelsen om, at det er svært at primtalsfaktorisere et stort tal. Talene anvendt ligger typisk i intervallet 2^{1024} til 2^{4096}. Hvilket svarer til tal mellem $\approx 1.798 \cdot 10^{308}$ og $\approx 1.044 \cdot 10^{1233}$ i decimaltal.[Caz14, s. 21]

1 Talteori

Talteorien beskæftiger sig med alle de hele tal $\mathbb{Z}$. Vi skal i løbet af dette afsnit kigge på hvordan talteori ligger til grunde for, at RSA virker i praksis.
Det er ikke alle definitioner og sætninger der bliver bevist, da det ville tage for lang tid. Alle beviser kan naturligvis findes i *Kryptografi – fra viden til videnskab.* [LN97]

1.1 Division med rest og divisor

I stedet for at regne brøker som rationelle mængder $\mathbb{Q}$ eller decimaltal, skal vi beskæftige os med hvordan man regner med rester. Det vil altså sige at vi udelukkende vil beskæftige os med heltal $\mathbb{Z}$.
Et tal $a \neq 0$ kan enten gå op i b eller ikke gå op i b. Vi definerer at a går op i b, hvis der ingen rest er, efter division. Eller sagt på en anden måde:

Definition 1.1

Givet tallet $a \neq 0$ og tallet $b \geq a$, vil a gå op i b, hvis der findes et heltal q, som opfylder $b = q \cdot a$.
Det skrives $a \mid b$, som betyder at a går op i b.
a betenes som *divisoren* eller en *faktor*, mens q kaldes for *kvotienten.*[LN97, s. 70]

Er der i stedet en rest efter division, vil man kunne skrive tallet som $a \nmid b$, som betyder at a ikke går op i b.

Sætning 1.2

Givet tallet $a > 0$ og tallet b, findes der specifikke tal q og r, som opfylder $b = q \cdot a + r$, hvor $0 \leq r < q$. r betegnes som *resten.*

Det ses ved nogle eksempler

Eksempel 1.3

Lad $a = 48$ og $b = 8$, det ses da at $q = 6$ og $r = 0$, da $48 = 6 \cdot 8 + 0$ og da $r = 0$, medfører det at $a \mid b$.

Lad $a = 17$ og $b = 4$, det ses da at $q = 4$ og $r = 1$, da $17 = 4 \cdot 4 + 1$ og da $r \neq 0$, medfører det at $17 \nmid 4$.

Man kan også udregne resten ud fra et decimaltal. Trykker man $17 \div 4$ ind på lommeregneren får man 4.25, man ganger så decimal delen med b, vil man få resten efter divisionen. I dette tilfælde får man $r = 0.25 \cdot 4 = 1$.

Sætning 1.4

Givet tallene $a, b, c \in \mathbb{Z}$, hvor $a, c \neq 0$, vil $a \mid b$ medføre at $ac \mid bc$.

Bevis. Beviset tager selvfølgelig udgangspunkt i definition 1.1. Da $a \mid b \implies b = q \cdot a$, vil man ved at gange c ind, på begge sider, blot få $bc = q \cdot ac$. Og da c indgår på begge sider af lighedstegnet, kan dette fjernes fra udtrkket, hvorved man er tilbage ved $b = q \cdot a$. □

Sætning 1.5

Givet tallene $a, b, c \in \mathbb{Z}$, hvor $a, c \neq 0$, vil $a \mid b \wedge b \mid c$ medføre at $a \mid c$.

Bevis. $a \mid b$ skrives som $b = q_1 \cdot a$, mens $b \mid c$ skrives som $c = q_2 \cdot b$. Ved at substituere b med udtyrkket fra $a \mid b$, haves $a = q_1 \cdot q_2 \cdot c$. $a \mid c$ er altså sand hvis kvotienten er $q_1 \cdot q_2$. □

Eksempel 1.6

Her skal gerne være lidt eksempler for at spice det lidt op...

1.1.1 Modulo

Når man laver division med heltal, vil man ofte kigge på resten efter division. Derfor er der naturligvis lavet en matematisk operator til at udregne rest efter division – den kaldes for *modulo*. Skriver man f.eks. $a \pmod{n}$, udregner man resten efter division, som angivet i sætning 1.2.

Definition 1.7

For $m \in \mathbb{Z}$ og $n \in \mathbb{N}$ gælder:

$$n \mid m \iff m \pmod{n} = 0$$

$$r = n - qm = m \pmod{n}$$

[LN97, s. 72]

Eksempel 1.8

Et par eksempler på brugen af modulo.

$$43 \pmod{6} = 1 \qquad 74 \pmod{10} = 4 \qquad 58 \pmod{6} = 4$$

1.2 Fælles divisor

En fællesvidisor bruges til at sige noget om sammenhængen mellem 2 tal.

Definition 1.9

For tallene $a, b, d \in \mathbb{Z}$, betegnes d som en *fælles divisor*, hvis $d \mid a \wedge d \mid b$.

a og b vil have et endeligt antal fælles divisorer, da et tal højere end enten a eller b naturligvis ikke er en fælles divisor (medmindre a og b begge er 0. Det kan altså konkluderes at der findes en største fælles divisor.

Definition 1.10

Den største fælles divisor for tallene a og b, betegnes som (a, b).

Eksempel 1.11

Lad $a = 8$ og $b = 12$.
Det ses at divisorne i 8 er: $\pm 1, \pm 2, \pm 4, \pm 8$.
Og det ses at divisorne i 12 er: $\pm 1, \pm 2, \pm 3, \pm 4, \pm 6, \pm 12$.
Til fælles har de altså divisorne: $\pm 1, \pm 2, \pm 4$.
Dermed bliver $(8, 12) = 4$.

Det ses at fortegnet for både a og b er ubetydeligt, da kvotienterne kan være negative.
Som vist, kan være bøvlet at udregne fælles divisorer for 2 tal, derfor kommer nu en sætning, til at mindske vanskeligheden lidt.

Sætning 1.12

$$(a, b) = (a, b \pmod a)$$

Bevis. Måske... □

1.2.1 Euclids algoritme

Euclids algoritme er essentielt gentagen anvendelse af sætning 1.12.

Definition 1.13: Euclids algoritme

Lad $a, b \in \mathbb{N}$, hvor $a \geq b$. Hvis $a \mid b$, så er $(a, b) = b$. Hvis $a \nmid b$, bruges følgende algoritme:

$$\begin{aligned} a &= bq_0 + r_0 && 0 < r_0 < b \\ b &= r_0 q_1 + r_1 && 0 \leq r_1 < r_0 \\ r_0 &= r_1 q_2 + r_2 && 0 \leq r_2 < r_1 \\ r_1 &= r_2 q_3 + r_3 && 0 \leq r_3 < r_2 \\ &\vdots \end{aligned}$$

Denne proces slutter, idet der findes en rest på 0. Det vil ske efter et endeligt antal gennemløb t.

$$\begin{aligned} r_{t-2} &= r_{t-1} q_t + r_t && 0 \leq r_t < r_{t-1} \\ r_{t-1} &= r_t q_{t+1} + r_0 \end{aligned}$$

Den sidste rest, som ikke er 0, er dermed den største fælles divisor.[Hun97, s. 11]

Det ses hvordan Euclids algoritme bruges, til at overskue udregningen af største fælles divisor.

Eksempel 1.14

$$\begin{array}{lllrlrr}
(1048, 672) & = (376, 672) & \text{idet} & 1048 & = 1 \cdot 672 & + & 376 \\
& = (376, 296) & - & 672 & = 1 \cdot 376 & + & 296 \\
& = (80, 296) & - & 376 & = 1 \cdot 296 & + & 80 \\
& = (80, 56) & - & 296 & = 3 \cdot 80 & + & 56 \\
& = (24, 56) & - & 80 & = 1 \cdot 56 & + & 24 \\
& = (24, 8) & - & 56 & = 2 \cdot 24 & + & 8 \\
& = (0, 8) & - & 24 & = 3 \cdot 8 & + & 0 \\
& = 8 & & & & &
\end{array}$$

Ovenstående metode er faktisk en algoritme, fremskrevet af Euclid. Af samme grund kaldes den Euclids algoritme. Da der allerede er et eksempel på den, vil algoritmen ikke vises – dog ligger den implementeret i Python som bilag.

1.3 Primtal

Hvad har primtal med det hele at gøre?

1.4 Eulers ϕ-funktion

Eulers phi funktion

2 Hashing

Hashing er en metode, brugt til at omdanne en tekststreng til en anden tekststreng, med en fikseret længde. Den nye tekststreng vil være pseudorandom - den vil fremstå tilfældigt, men faktisk være regelmæssig, baseret på inputtet. Til forskel fra RSA skal man ikke skal generere nogle tal førend metoden bruges, hvilket medfører at der altid bliver genereret det samme output, ved det samme input.
Hashing en også envejsfunktion, modsat RSA, hvor man jo både kan kryptere og dekryptere data. Med hashing kan man altså ikke gå baglæns, hvis man kun har et hash af en tekst.

2.1 Fordelen ved hashing

Det at hashing kun går en vej, virker måske lidt nytteløst, da man ikke længere er i stand til at finde ud af hvad man har hashet. I stedet bruger man hashing til at gemme kodeord, sådan at hvis nogle kodeord bliver lækket, er de stadig beskyttede. Dette kan lade sig gøre, da hashing altid giver det samme hash ved brug af samme input. Det vil altså sige at hvis man hasher *abc*, vil man få det samme output hver gang. Da et hash er pseudorandom, vil det ikke sige noget om hvad det tidligere har været - der er som sådan ikke noget mønster i metoden. Det ses i eksempel 2.1.

Eksempel 2.1

Det ses at, på trods af lignende input, er hashet helt forskelligt.

```
MD5(abc) = 900150983cd24fb0d6963f7d28e17f72
MD5(Abc) = 35593b7ce5020eae3ca68fd5b6f3e031
MD5(abd) = 4911e516e5aa21d327512e0c8b197616
```

Måden man anvender dette på, er at hashe et kodeord inden det bliver puttet ind i databasen. Når så brugeren forsøger at logge ind, hasher man igen det indtastede kodeord og hvis det stemmer over ens, med det liggende i databasen, kan man formode at brugeren har indtastet det rigtige kodeord.

Der findes mange forskellige kryptografiske hashfunktioner, blandt de mere kendte er `MD5`, `SHA256` og `SHA512`. Essentielt er de opbygget på samme måde, forskellen i dem er hvor langsomme og dermed sikre de er. For at forstå hvorfor en langsom hashfunktion er god, er det vigtigt at forstå essensen i en hashfunktion. Det at den kun kan bruges en vej, gør nemlig at hvis man gerne vil finde ud af den originale tekst til et hash, er man nødt til at prøve sig frem, i IT-verdenen kaldes dette for *brute-force*.

Eksempel 2.2 (Kompleksiteten ved brute-force)

Antag at man har et hash, hashet af `MD5`, hvor man gerne vil finde den oprindelige tekst. Man ved at teksten består af 8 tegn – som kan være både store og små bogstaver samt tal, men ingen symboler. I det danske alfabet er der 29 bogstaver og disse kan både fremstå stor og små $2 \cdot 29 = 58$, derudover er der 10 forskellige

tal at tage hensyn til $58 + 10 = 68$. Da alle tegn har mulighed for at stå på alle 8 pladser, er der nu $68^8 \approx 4.572 \cdot 10^{14}$ muligheder. En moderne computer kan udregne omkring $4 \cdot 10^9$ hashes i sekundet, i `MD5`. Mens en lille server kan yde omkring 10 gange så meget $4 \cdot 10^{10}$.[Pou16] Det udregnes hvor mange sekunder det vil tage at hashe alle kombinationer af 8 bogstaver og tal, for henholdsvis en computer og en lille server.

$$\frac{4.572 \cdot 10^{14}}{4 \cdot 10^9} = 114300 \qquad \frac{4.572 \cdot 10^{14}}{4 \cdot 10^{10}} = 11430$$

Det omskrives til timer.

$$\frac{114300}{60 \cdot 60} = 31.75 \qquad \frac{11430}{60 \cdot 60} = 3.175$$

Det kan altså konkluderes at det er klogt at vælge et kodeord på mere end 8 tegn. Samt at en god hashfunktion tager lang tid, hvorved det forstås at funktioner som `SHA256` og `SHA512` er at foretrække over `MD5`.

2.2 Rainbow tables og Salt

Der findes en *løsning* til problemet, om at hashes er svære at gendanne, er der opfundet det man kalder *rainbow tables*. Et rainbow table er en tabel, hvori der står en masse hashes samt hvad de er oprindet af. Det vil altså sige at man ved at lave et opslag i et rainbow table, meget hurtigt kan finde frem til om et hash allerede er blevet de-hashet. Et rainbow table indeholder naturligvis ikke alle hashes – det ville være en uoverkommelig opgave at liste dem alle. Det er altså kun de hashes – og dermed også kodeord, som er mest anvendte, der ligger i et rainbow table. Det er af samme grund, at man ikke blot skal bruge koden `password`, da den helt sikkert ligger i et rainbow table.

Der er dog heldigvis opfundet en løsning til bekæmpelse af brugen af rainbow tables og den kaldes for salt. Pointen med et salt er at skabe rod i hashet.

3 Forskkel på hashing og kryptografi

Hvorfor man ikke blot kan nøjes med den ene

3.1 Alice & Bob

Eksempel på brug af begge dele

4 Hvorfor RSA er vigtig

Hvor bliver det brugt

4.1 En verden uden RSA

Litteratur

[Caz14] Cazimi, Blerim. *RSA - KRYPTOSYSTEMET*. Dec. 2014. URL: http://www.frividen.dk/wp-content/uploads/SRP-HTX-MatematikA-InformationsteknologiB-12-tal-RSA-kryptering.pdf.

[Hun97] Hungerford, Thomas W. *Abstract Algebra, an indtroduction*. Engelsk. second edition. Brooks/Cole, 1997, s. 11–12. ISBN: 0-03-010559-5.

[LN97] Landrock, Peter og Nissen, Knud. *Kryptologi - fra viden til videnskab*. Dansk. 1. udg. Abacus, 1997. ISBN: 87-89182-62-6.

[Pou16] Pound, Mike. *Password Cracking - Computerphile*. YouTube. Jul. 2016. URL: https://www.youtube.com/watch?v=7U-RbOKanYs.

[Ves07] Vestergaard, Erik. *RSA-kryptosystemet*. 2007. URL: https://matematikfysik.dk/mat/noter_tillaeg/RSA.pdf.