Public-key kryptografi med RSA
02621 Numeriske Algoritmer

Thomas Jakobsen og Lars R. Knudsen*

Juni 2003

Kryptografi handler bl.a. om hemmeligholdelse og om at sikre autenticiteten af data. I dette projekt skal kryptosystemet RSA undersøges og implementeres. RSA er et såkaldt public-key-system, som har en række fordele i forhold til de klassiske kryptosystemer. RSA er et af de oftest benyttede public-key-systemer og det bruges f.eks. i forskellige internet-browsere, i dankort-automater og i krypteringsprogrammet PGP.

Public-key-systemer benytter to nøgler, en til at kryptere med og en anden til dekryptering. Det betyder at parterne, som skal kommunikere, ikke behøver at udveksle en fælles hemmelig nøgle via en sikker kanal før kommunikationen kan begynde. RSA-systemet kan også benyttes til at frembringe digitale signaturer, som kan sikre et dokuments autenticitet.

Der er lagt op til, at implementeringen foregår i C++. Da RSA bygger på matematik, som involverer store tal kan man med fordel benytte biblioteket GMP, Gnu Multiprecision Library, som er til rådighed på projektets hjemmeside. Det er ikke et krav, at I benytter C++ eller GMP, men der kan desværre kun ydes hjælp i begrænset omfang til andre metoder (f.eks. Java m. Bignum-pakken eller Maple).

I vil få udleveret et RSA-nøglesæt ved projektets start. Det er meningen, at nøglerne skal benyttes, når rapporten skal afleveres. Rapporten skal v.h.a. det implementerede RSA-system underskrives digitalt inden aflevering.

1 Introduktion til kryptografi

Kryptologi er en ældgammel disciplin, som går flere tusinde år tilbage i tiden. I dag omfatter disciplinen mange forskelligartede ting, heriblandt kryptering. Kryptering er kunsten at holde data hemmelige for uvedkommende. I kryptering betragter man oftest en situation, hvor en part, kaldet Alice, vil sende en meddelelse til en anden part, kaldet Bob, således at ingen tredje part skal kunne lære indholdet af meddelelsen.

I den klassiske kryptering, også kaldet "secret-key" kryptering, forudsættes det, at Alice og Bob deler noget information, som kun de to har kendskab til. Se Figur 1. Vi siger, at de først skal *udveksle en hemmelig nøgle*, k. Med denne nøgle kan Alice kryptere sin meddelelse x og sende resultatet y til Bob. Hvis e betegner krypteringsfunktionen, kan vi opskrive det påfølgende vis:

$$y = e_k(x). \tag{1}$$

Bob kan herefter bruge nøglen til at *dekryptere* den sendte, krypterede meddelelse:

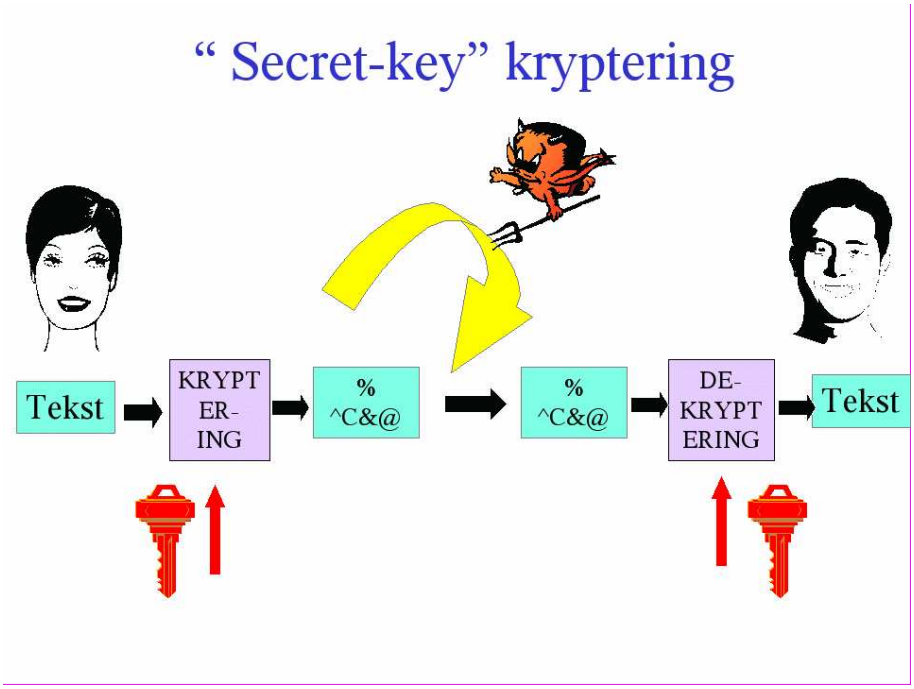
$$x = d_k(y) = e_k^{-1}(y) \tag{2}$$

Her betegner d dekrypteringsfunktionen, som i dette tilfælde er den inverse funktion til krypteringsfunktionen e. En ikke-krypteret meddelelse kaldes normalt en klartekst, og en krypteret meddelelse en chiffertekst.

Hvis Alice og Bob ikke fortæller andre, hvad (værdien af) nøglen er, og hvis ellers krypteringssystemet er sikkert, så kan Alice regne med, at det kun er Bob, der kan læse den originale meddelelse.

I 1976 fandt Diffie og Hellman fra USA på at dele krypteringsnøglen i to dele, en offentlig del, k_p, og en hemmelig del, k_s. Se Figur 2. Bob vælger en nøgle og sender den offentlige del, k_p til Alice. Nu kan Alice

*thomas@mat.dtu.dk, knudsen@mat.dtu.dk



Figur 1: Klassisk kryptering.

sende en krypteret klartekst til Bob ved hjælp af den offentlige nøgle

$$y = e_{k_p}(x) \tag{3}$$

og Bob kan dekryptere chifferteksten ved hjælp af sin hemmelige nøgle

$$x = d_{k_s}(y). \tag{4}$$

Hvis dekryptering skal give mening, skal det selvfølgelig gælde, at $d_{k_s}(e_{k_p}(x)) = x$. Hvis Bob ikke fortæller nogen, hvad den hemmelige nøgle er, så kan Alice regne med, hvis ellers krypteringssystemet er sikkert, at det kun er Bob, der kan dekryptere chifferteksten. Sådan et system kaldes et *public-key* (krypterings)system og det smarte er, at Alice og Bob ikke behøver at mødes eller udveksle en hemmelig nøgle, før de begynder med at sende hinanden krypterede meddelelser.

RSA er et eksempel på et public-key-system. Systemet blev opfundet i 1977 af Rivest, Shamir og Adleman. Inden vi skal høre om RSA, er der lidt grundlæggende matematik, der skal forklares først.

2 Primtal og modulær aritmetik

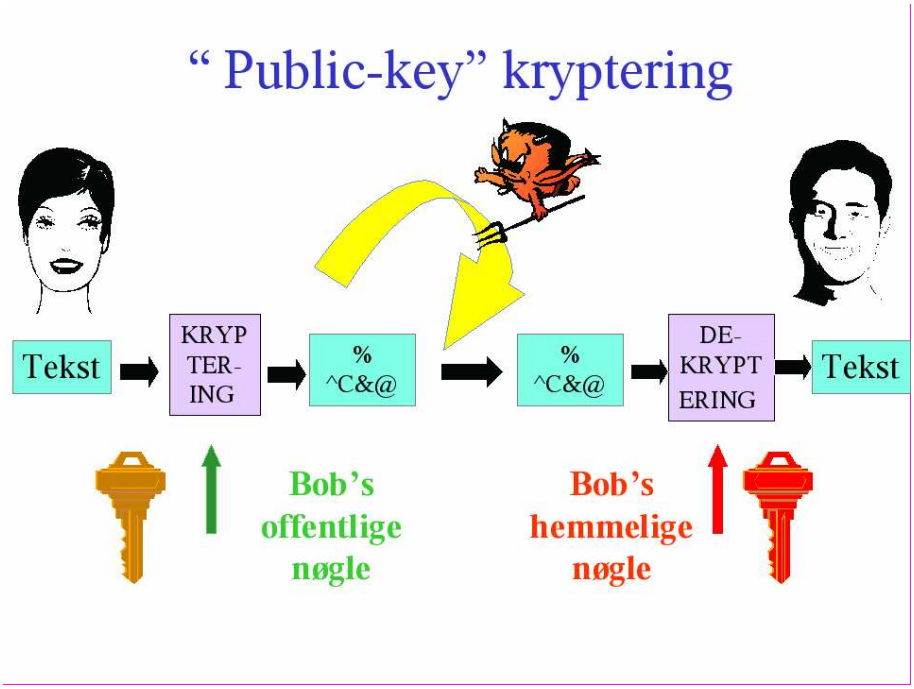
Definition 1 *Et heltal $p \geq 2$ siges at være et* primtal, *hvis tallets eneste positive heltalsfaktorer er* 1 *og p. Ellers kaldes tallet* sammensat.

Eksempel. Tallene 2, 3, 5, 7, 11, 13 og 41 alle primtal, men 12, 25 og 91 er sammensatte.

Definition 2 *Lad x og y være heltal og lad n være et positivt heltal. Vi siger, at x og y er kongruente modulo n, hvis resten ved division af x med n er lig med resten ved division af y med n. Vi skriver $x \equiv y \pmod{n}$. Med andre ord gælder det, at $x \equiv y \pmod{n}$, hvis n går op i $x - y$.*

Eksempel. $15 \equiv 7 \bmod 4$ og 4 går op i 15-7.

1. Lad $n = 13$, $a = 12345$ og $b = 103579$. Verficér om $a \equiv b \pmod{n}$.
   Lad $n = 12$, $a = 123$. Find et heltal b så $a \equiv (b+1) \pmod{n}$.



Figur 2: Public-key kryptering.

Vi skriver “a mod n” (hvor mod n er uden parenteser) for det tal b, hvorom der gælder, at $b \equiv a \pmod{n}$ og $0 \leq b < n$. Vi siger, at b er “a reduceret modulo n”. Der er nogle simple regler, man kan bruge til at gøre sådanne udregninger lettere. F.eks. er udtrykket $(b+c)$ mod n det samme som udtrykket $((b \bmod n)+(c \bmod n)) \bmod n$, og udtrykket (bc) mod n er det samme som udtrykket $((b \bmod n)(c \bmod n)) \bmod n$.

Eksempel. $2 = 17 \bmod 3$. $9 \cdot 7 \bmod 8$ er 63 reduceret modulo 8, altså 7. Dette kan også udregnes som flg.: $(9 \bmod 8) \cdot 7 \bmod 8$ som jo også er 7.

2. Lad $n = 191$, $a = 112$ og $b = 203$. Find ab mod n.
Lad $n = 117$, $a = 17$ og $b \equiv 10 \pmod{n}$. Find ab mod n.

Definition 3 *Den største fælles divisor af to heltal, a og b, er det største heltal, som går op i både a og b. Vi skriver* $\text{sfd}(a,b)$.

Eksempel. $\text{sfd}(21,98) = 7$.

3. Lav et (Maple) program som tager et positivt heltal n som input og som returnerer $\text{sfd}(a,n)$ for alle a, hvor $0 < a < n$.

Definition 4 *Lad n være et positivt heltal, og lad* $x < n$ *være et ikke-negativt heltal. Vi siger, at y er den* multiplikative invers *til x modulo n, hvis* $1 = (xy) \bmod n$. *Vi bruger også notationen* $x^{-1} \bmod n$ *for y.*

Sætning 1 *Lad n være et positivt heltal, og lad* $x < n$ *være et heltal. Så har x en multiplikativ invers modulo n, hvis og kun hvis* $\text{sfd}(x,n) = 1$.

Eksempel. 112 er den multiplikative inverse til 7 mod 261, da $7 \cdot 112 = 784 \equiv 1 \bmod 261$. 3 har **ikke** nogen multiplikativ invers modulo 261, da $\text{sfd}(3,261) = 3 \neq 1$, dvs. for alle heltal z gælder, at $3z \not\equiv 1 \bmod 261$.

4. Lav et (Maple) program, som tager et heltal n som input, og som for ethvert heltal a mellem 0 og n gør følgende: hvis a har en multiplikativ invers, udskrives a og den inverse, ellers udskrives intet. Udskriv resultatet af programmet for $n = 41$ og $n = 24$.

3 RSA-systemet

RSA fungerer på følgende måde.

3.1 Opsætning

1. Find to store, tilfældige, forskellige primtal p og q af ca. samme størrelse.
2. Beregn $n = pq$ og $\Phi = (p-1)(q-1)$.
3. Vælg et heltal a, hvor $1 < a < \Phi$ og $\text{sfd}(a, \Phi) = 1$.
4. Find den inverse til a modulo Φ, $b \equiv a^{-1} \pmod{\Phi}$.
5. A's offentlige nøgle er $k_p = (n, b)$; A's private nøgle er $k_s = (a, p, q)$.

Med "stor" menes tal på hundredevis af cifre. Typisk benyttes enten 1024 bit eller 2048 bit svarende til henholdsvis over 300 og over 600 cifre.

3.2 Kryptering

Beskeden repræsentes som et tal x, $0 \leq x < n$. Chifferteksten y findes ved udregningen

$$y = e_{k_p}(x) = x^b \bmod n \qquad \text{(kryptering)} \tag{5}$$

Om lidt skal vi se, hvordan dette udtryk kan beregnes på en effektiv måde.

Eksempel. Lad $p = 61, q = 71$. Da er $n = pq = 4331$ og $\Phi = 4200$. Vælg f.eks. $a = 11$ og bemærk at $\text{sfd}(11, 4200) = 1$. Da bliver b beregnet til 2291. Kryptering af beskeden $x = 199$ er da $y = 199^{2291} \bmod 4331 = 696$. (Størrelsen på tallene i eksemplet er ikke realistiske, hvis sikkerheden skal være høj.)

5. Lav et Maple-program, som RSA-krypterer følgende beskeder $x = 2$, $x = 61$ og $x = 1879$ med p, q, a, b som i foregående eksempel.

3.3 Dekryptering

Dekrypteringsprocessen minder om kryptering, men den hemmelige nøgle indgår i stedet for den offentlige. Den modtagne chiffertekst y dekrypteres på følgende vis:

$$x = d_{k_s}(y) = y^a \bmod n \qquad \text{(dekryptering)} \tag{6}$$

Systemet ville ikke være meget værd, hvis denne operation ikke gendannede klarteksten. Det gør den heldigvis:

Sætning 2 *RSA-dekryptering gendanner klarteksten, d.v.s.* $d_{k_s}(e_{k_p}(x)) = x$.

6. Dekrypter de krypterede beskeder fra foregående opgave og verificer at de oprindelige beskeder genopstår.

7. (frivillig) Bevis ovenstående sætning. D.v.s. vis sidste lighedstegn i følgende udtryk:

$$\begin{aligned} d_{k_s}(e_{k_p}(x)) &\equiv (x^b \bmod n)^a \bmod n \pmod{n} \\ &\equiv x^{ba} \pmod{n} \\ &\equiv x \end{aligned}$$

Du skal bruge et resultat fra algebraen, Lagrange's sætning, som siger, at $x^{\Phi} = 1 \pmod{n}$. Vink: Udnyt at b er a's inverse modulo Φ.

3.4 Sikkerhed

Hvis en tredjepart kan faktorisere n, så kan han finde p og q, og dermed beregne Φ og følgelig den hemmelige nøgle a ud fra b. Dette vil bryde sikkerheden i systemet. Derfor skal p og q vælges så store at det bliver for svært at faktorisere n. Man siger, at RSA-systemets sikkerhed baseres på hårdheden af faktoriseringsproblemet.

Firmaet RSA Security, som indtil for nyligt havde patent på systemet (det er nu udløbet), har på deres hjemmeside en konkurrence, hvor man kan vinde helt op til $200.000, hvis man faktoriserer tilpas store (RSA-)tal. Se
http://www.rsasecurity.com/rsalabs/challenges/factoring .

3.5 Digital signatur

Ved digital signatur forstås et system, som kan benyttes til at fastlægge autenticiteten af et dokument. F.eks. kan Bob ved modtagelse af en besked fra Alice sikre sig, at den virkelig kommer fra hende. RSA-systemet kan benyttes til digital signatur på følgende vis:

Når Alice sender beskeden x til Bob benytter hun sin egen private nøgle k_s og udregner

$$y = d_{k_s}(x) \qquad \text{(signering)}. \tag{7}$$

Hun sender herefter (x,y) til Bob som herefter undersøger om

$$x = e_{k_p}(y) \qquad \text{(verifikation)}, \tag{8}$$

hvor k_p er Alice's offentlige nøgle. Hvis ligningen holder, så kan Bob gå ud fra, at afsenderen virkelig er Alice. Hun er nemlig den eneste, der kan beregne y, fordi det kun er hende, som kender den hemmelige nøgle k_s. Omvendt kan alle undersøge autenticiteten af den digitale signatur, for alle har adgang til Alice's offentlige nøgle.

8. Lad $n = 11413$, $a = 7467$, $b = 3$. Undersøg om y er en gyldig signatur for x for følgende par (x,y): $(100,4443)$, $(200,7594)$, $(4746,4746)$ og $(4747,4747)$.

4 Beregning

4.1 Multiplikation

En grundliggende operation i RSA er multiplikation modulo et meget stort tal n. I dette projekt går vi ikke i detaljer med, hvorledes man rent faktisk udregner disse produkter, da vi benytter allerede implementerede algoritmer.

Kort kan det nævnes, at de simpleste måder blot er generaliseringer af almindelige "folkeskolemetoder" til multiplikation af tal. Se også afsnittet om implementering.

4.2 Modulær eksponentiering

Under kryptering skal man udregne potensopløftningen $y = x^a \bmod n$. Spørgsmålet er, hvorledes dette kan gøres på en effektiv vis. Hvis f.eks. $a = 3$ kan man finde y ved at multiplicere x med sig selv et passende antal gange (modulo n), $y = x \cdot x \cdot x \bmod n$. Det kræver i alt to multiplikationer, og den operation kender vi allerede.

Men hvis a nu er et tal med hundredevis af cifre er det en meget langsom og dårlig metode. I stedet benytter man sig af den såkaldte "kvadrer-og-multiplicer"-metode (engelsk: "square-and-multiply").

Metoden kan illustreres ved eksemplet $x^{17} = (((x^2)^2)^2)^2 \cdot x$. Kvadrering kræver en enkelt multiplikation (mod n), så derfor kan x^{17} altså beregnes v.h.a. i alt fem multiplikationer (i stedet for 16 som ved den langsomme metode).

Eksemplet kan generaliseres: Lad $k_t, k_{t-1}, \ldots, k_0$ være den binære repræsentation af k, d.v.s. $k = \sum_{i=0}^{t} a^{k_i} 2^i$. Så gælder følgende sammenhæng:

$$a^k = \prod_{i=0}^{t} a^{k_i 2^i} = ((((a^{k_t})^2) a^{k_t - 1})^2) \cdots) a^{k_1})^2 a^{k_0} \tag{9}$$

Ligningen kan benyttes som grundlag til en effektiv algoritme til modulær eksponentiering:

Algoritme MOD_EXP. Modulær eksponentiering i Z_n via kvadrer-og-multiplicer-metoden.

INPUT: $x \in Z_n$, og et heltal k, $0 \leq k < n$..

OUPUT: x^k mod n.

1. Sæt $s = 1$.

2. Sæt $X = x$.

3. Sålænge $k \neq 0$, gentag:

3.1 Hvis k mod $2 = 1$, så sæt $s = (X \cdot s)$ mod n.

3.2 Sæt $k = \lfloor k/2 \rfloor$.

3.3 Sæt $X = X^2$ mod n.

4. Returner s.

4.3 Udregning af sfd

Ved opsætning af RSA-systemet skal man beregne største fælles divisor, sfd, af to tal. Dette kan gøres v.h.a. *Euklids algoritme*:

ALGORITME. EUKLID.

Input: To heltal i og j, hvor $i \geq j$.

Ouput: sfd(i, j), største fælles divisor af i og j.

1. Sålænge $j \neq 0$, gør følgende

1.1 Sæt $r = i$ mod j

1.2 Sæt $i = j$

1.3 Sæt $j = r$

2. Returner a.

9. Udregn sfd$(123, 864)$ i hånden.

4.4 Udregning af modulære inverse

Hvis man har et a, så sfd$(a, n) = 1$ kan man finde b, sådan at $ab \equiv 1 \pmod{n}$. Man kan med andre ord finde a's inverse modulo n.

Til dette kan man benytte *Euklids* udvidede *algoritme*:

ALGORITME UDV_EUKLID(i, j).

INPUT: to heltal i og j, hvor $i \geq j$.

OUTPUT: $d =$ sfd(i, j) og heltal s og t som opfylder $is + jt = d$.

1. Hvis $j = 0$ så saet $d = a$, $s = 1$, $t = 0$ og returner (d,s,t)

2. Sæt s_2=1, $s_1 = 0$, $t_2 = 0$, $t_1 = 1$.

3. Sålænge $b > 0$ gør følgende:

3.1 Sæt $q = \lfloor i/j \rfloor$, $r = i - qj$, $s = s_2 - qs_1$, $t = t_2 - qt_1$.

3.2 Sæt $i = j$, $j = r$, $s_2 = s_1$, $s_1 = s$, $t_2 = t_1$, $t_1 = t$.

4. Sæt $d = a$, $s = s_2$, $t = t_2$ og returner (d,x,y).

Som beskrevet returnerer den udvidede algoritme yderligere to værdier, s og t, som opfylder $is + jt = d$. Dette kan udnyttes til at finde modulære inverse.

Sætning 3 *Hvis* $sfd(a,\Phi) = 1$, *da returnerer UDV_EUKLID*(a,Φ) *værdien* $s \equiv a^{-1} \pmod{\Phi}$.

Når vi skal finde b under opsætning af RSA-systemet kan vi altså benytte Euklids udvidede algoritme.

5 Implementering

De medfødte datatyper i C++ og Java kan kun håndtere 32 bit eller 64 bit tal. Det er ikke nok til en sikker implementering af RSA. Implementeringer af RSA må man derfor selv sørge for at kunne håndtere store tal.

Som tidligere nævnt tager denne projektbeskrivelse udgangspunkt i en implementering i C++ v.h.a. biblioteket GMP. Det er i orden at benytte andre tilgange (f.eks. Java og biblioteket Bignum), men så må I selv håndtere eventuelle tekniske problemer.

I GMP benyttes datatypen mpz_class til variable som repræsenterer store heltal. GMP benytter oftest overloadede C++operatorer, hvilket medfører en høj læsbarhed. Enkelte bit i et tal k kan findes ved at benytte funktionen
mpz_testbit(k.get_mpz_t(), i)
hvor i er bittens nummer. Til implementeringen af modulær eksponentiering kan denne funktion benyttes med fordel i stedet for direkte at udregne k mod 2 (som er langsommere).

Følgende programeksempel benytter de fleste af de GMP-funktioner, som I får brug for.

```
#include <iostream>
#include <gmpxx.h>
using namespace std;

int main()
{
  mpz_class a;
  a="9876543210";
  mpz_class b;
  b="1234567890";
  mpz_class prod=a*b;
  mpz_class div=a/b;
  mpz_class mod=a%b;

  cout << "a=" << a << endl;
  cout << "b=" << b << endl;
  cout << "a*b=" << prod << endl;
  cout << "a/b=" << div << endl;
  cout << "a mod b=" << mod << endl;
  cout << "a's least significant bit: "
       << mpz_tstbit(a.get_mpz_t(), 0) << endl;
}
```

Gemmes programmet som rsa.cpp kan det linkes med GMP-biblioteket ved at udføre Unix-kommandoen gcc rsa.cpp -l ntl.lib

Køres programmet ses følgende output:

```
a=9876543210
b=1234567890
a*b=12193263111263526900
a/b=8
a mod b=90
a's least significant bit: 0
```

GMP indeholder også funktioner til modulær eksponentiering og Euklids (udvidede) algoritme. *Det er ikke meningen at disse skal benyttes*, i stedet skal I selv implementere funktionerne.

En udgave af GMP, som allerede er compilet og som kan køre på databarens maskiner findes på adressen http://www.mat.dtu.dk/persons/T.Jakobsen/02621. Den fulde dokumentation til GMP findes også her.

10. Implementer modulær eksponentiering ved hjælp af kvadrer-og-multiplicer-metoden.

11. Beregn 113^{549} mod 107311 v.h.a. den implementerede MOD_EXP-funktion.

12. Implementer Euklids algoritme for største fælles divisor.

13. Udregn sfd$(983,1985)$ og sfd$(330048,163296)$ v.h.a. Euklids algoritme. Udskriv mellemregningerne.

14. Implementer Euklids udvidede algoritme for modulære inverse.

15. Benyt Euklids udvidede algoritme til at finde b hvis $a = 123$, $\Phi = 18467$. Medtag mellemregninger. Verificer at b vitterligt er det er den inverse ved at beregne ab mod Φ og se om det giver 1.

16. Vis Sætning 3.

17. Implementer RSA v.h.a. de allerede udviklede funktioner MOD_EXP og UDV_EUKLID. Der skal implementeres såvel opsætning, kryptering og dekryptering. I opsætningstrinet kan I gå ud fra, at primtallene er til rådighed allerede, eller I kan benytte de indbyggede primtalstest som findes i GMP.

6 Aflevering

Inden projektet afleveres, skal det underskrives elektronisk. Dette foregår ved at samle alle projektets dokumenter i en enkelt fil (f.eks. v.h.a. zip eller tar) og herefter underskrive denne fil v.h.a. det udviklede RSA-program og den udleverede nøgle.

Da RSA-systemet kun kan underskrive tal mellem 1 og n benytter man sig oftest af en såkaldt (kryptografisk) hash-funktion. En hash-funktion tager tal af vilkårlig længde og afbilder dem på et lukket heltalsinterval, f.eks. mængden af 128 bit eller 160 bit heltal. I projektet her benyttes hash-funktionen MD5. Det er ikke meningen at MD5 skal implementeres, i stedet skal man benytte programmet md5.exe til Windows eller md5 til Unix, som ligger til download på adressen
http://www.mat.dtu.dk/persons/T.Jakobsen/02621.

Programmet md5 genererer et såkaldt fingeraftryk af den fil, hvis navn programmet får som argument. Fingeraftrykket er en hexadecimal repræsentation af filens hash-værdi. Det er denne værdi man underskriver.

18. Saml jeres rapport og alle opgaveresultaterne (undtagen dette) i et enkelt filarkiv, beregn MD5-fingeraftrykket af filen, konverter fingeraftrykket til decimaltal, og underskriv det v.h.a. jeres RSA-implementering og den udleverede nøgle. Aflever arkivet og det underskrevne fingeraftryk.