Kryptering

Kryptering

Scenarium:
Alice ønsker at sende en meddelelse (klartekst) til Bob
Kommunikationskanalen er usikker og kan blive aflyttet
Hvis Alice og Bob er blevet enige om et skema for kryptering, kan meddelelsen sendes krypteret (som kodetekst)

Spørgsmål:
- Hvad er et godt skema til kryptering?
- Hvilken kompleksitet har kryptering/afkryptering?
- Hvor lang er kodeteksten i forhold til klarteksten?
- Hvis Alice og Bob aldrig før har kommunikeret med hinanden, hvordan kan de så blive enige om et skema for krypteringen?

klartekst → krypter → kodetekst → afkrypter → klartekst

Traditionel kryptografi

Metoder til kryptering (kodeskrift) blev studeret allerede i oldtiden

Cæsars kodeskrift:
- erstat *a* med *d*
- erstat *b* med *e*
- ...
- erstat z med *c*

Cæsars kodeskrift er et eksempel på en **monoalfabetisk erstatningskode**, der permuterer de enkelte tegn i klarteksten

Statistiske angreb

Udstyret med simpel statistisk viden kan man let bryde en sådan kode:
- hyppigste bogstaver i engelsk tekst: e, t, o, a, n, i, ...
- hyppigste *digrammer* i engelsk tekst: th, in, er, re, an, ...
- hyppigste *trigrammer* i engelsk tekst: the, ing, and, ion, ...

Den første beskrivelse af dette angreb med frekvensanalyse forekommer i en bog, skrevet i det 9'nde århundrede af den arabiske filosof al-Kindi

Eksempel (S. Singh, The Code Book, 1999):

PCQ VMJYPD LBYK LYSO KBXBJXWXV BXV ZCJPO EYPD KBXBJYUXJ
LBJOO KCPK. CP LBO LBCMKXPV XPV IYJKL PYDBL, QBOP KBO BXV
OPVOV LBO LXRO CI SX'XJMI, KBO JCKO XPV EYKKOV LBO DJCMPV
ZOICJO BYS, KXUYPD: "DJOXL EYPD, ICJ X LBCMKXPV XPV CPO PYDBLK
Y BXNO ZOOP JOACMPLYPD LC UCM LBO IXZROK CI FXKL XDOK XPV LBO
RODOPVK CI XPAYOPL EYPDK. SXU Y SXEO KC ZCRV XK LC AJXNO X
IXNCMJ CI UCMJ SXGOKLU?" OFYRCDMO, LXROK IJCS LBO LBCMKXPV
XPV CPO PYDBLK

Frekvensanalyse (1)

Vi identificerer den hyppigste tegn, digrammer og trigrammer i klarteksten

Eksempel:

PCQ VMJYPD LBYK LYSO KBXBJXWXV BXV ZCJPO EYPD KBXBJYUXJ
LBJOO KCPK. CP LBO LBCMKXPV XPV IYJKL PYDBL, QBOP KBO BXV
OPVOV LBO LXRO CI SX'XJMI, KBO JCKO XPV EYKKOV LBO DJCMPV
ZOICJO BYS, KXUYPD: "DJOXL EYPD, ICJ X LBCMKXPV XPV CPO PYDBLK
Y BXNO ZOOP JOACMPLYPD LC UCM LBO IXZROK CI FXKL XDOK XPV LBO
RODOPVK CI XPAYOPL EYPDK. SXU Y SXEO KC ZCRV XK LC AJXNO X
IXNCMJ CI UCMJ SXGOKLU?" OFYRCDMO, LXROK IJCS LBO LBCMKXPV
XPV CPO PYDBLK

Første gæt:

LBO er THE

Frekvensanalyse (2)

Under antagelse af, at LBO repræsenter THE, erstatter vi L med T, B med H, og O med E, og får

PCQ VMJYPD THYK TYSE KHXHJXWXV HXV ZCJPE EYPD KHXHJYUXJ
THJEE KCPK. CP THE THCMKXPV XPV IYJKT PYDHT, QHEP KHO HXV
EPVEV THE LXRE CI SX'XJMI, KHE JCKE XPV EYKKEV THE DJCMPV
ZEICJE HYS, KXUYPD: "DJEXT EYPD, ICJ X THCMKXPV XPV CPE PYDHTK
Y HXNE ZEEP JEACMPTYPD TC UCM THE IXZREK CI FXKT XDEK XPV THE
REDEPVK CI XPAYEPT EYPDK. SXU Y SXEE KC ZCRV XK TC AJXNE X
IXNCMJ CI UCMJ SXGEKTU?"EFYRCDME, TXREK IJCS THE THCMKXPV
XPV CPE PYDBTK

Afkryptering

Kode:

X	Z	A	V	O	I	D	B	Y	G	E	R	S	P	C	F	H	J	K	L	M	N	Q	T	U	W
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

Kodetekst:

PCQ VMJYPD LBYK LYSO KBXBJXWXV BXV ZCJPO EYPD KBXBJYUXJ LBJOO KCPK. CP LBO LBCMKXPV XPV IYJKL PYDBL, QBOP KBO BXV OPVOV LBO LXRO CI SX'XJMI, KBO JCKO XPV EYKKOV LBO DJCMPV ZOICJO BYS, KXUYPD: "DJOXL EYPD, ICJ X LBCMKXPV XPV CPO PYDBLK Y BXNO ZOOP JOACMPLYPD LC UCM LBO IXZROK CI FXKL XDOK XPV LBO RODOPVK CI XPAYOPL EYPDK. SXU Y SXEO KC ZCRV XK LC AJXNO X IXNCMJ CI UCMJ SXGOKLU?" OFYRCDMO, LXROK IJCS LBO LBCMKXPV XPV CPO PYDBLK

Klartekst:

Now during this time Shahrazad had borne King Shahriyar three sons. On the thousand and first night, when she had ended the tale of Ma'aruf, she rose and kissed the ground before him, saying: "Great King, for a thousand and one nights I have been recounting to you the fables of past ages and the legends of ancient kings. May I make so bold as to crave a favour of your majesty?"
Epilogue, Tales from the Thousand and One Nights

Kryptering med hemmelig nøgle

En **hemmelig-nøgle** kode bruger en unik nøgle $\boldsymbol{K}$ til kryptering og afkryptering

Cæsars generaliserede kode bruger modulær addition af hvert tegn (betragtet som et heltal) med nøglen:

$\boldsymbol{C}[i] = (\boldsymbol{P}[i] + \boldsymbol{K}) \bmod \boldsymbol{m}$ (kryptering)
$\boldsymbol{P}[i] = (\boldsymbol{C}[i] - \boldsymbol{K}) \bmod \boldsymbol{m}$ (afkryptering)

Mere sikre hemmelig-nøgle krypteringskoder er blevet opfundet de seneste 50 år

Eksempler: DES 3DES IDEA BLOWFISH

Med hemmelig-nøgle kryptering må en særskilt nøgle etableres for ethvert par af deltagere

Offentlig-nøgle kryptering

Bob bruger et par af nøgler ($\boldsymbol{K_E}$,$\boldsymbol{K_D}$),
gør nøglen $\boldsymbol{K_E}$ offentlig og holder
nøglen $\boldsymbol{K_D}$ privat

Alle og enhver kan bruge den offentlige nøgle $\boldsymbol{K_E}$ til at kryptere en klartekst til Bob
Kun Bob kan afkryptere kodeteksten ved brug af sin private nøgle $\boldsymbol{K_D}$

Det mest populære krypteringsskema er RSA, opkaldt efter sine opfindere:
Rivest, Shamir og Adleman (1978)
RSA-patentet udløb 2000

offentlig nøgle ($\boldsymbol{K_E}$) ↓ privat nøgle ($\boldsymbol{K_D}$) ↓

klartekst → **krypter** → kodetekst → **afkrypter** → klartekst

Talteoretiske algoritmer

Fakta om tal

Primtal $\boldsymbol{p}$:
- $\boldsymbol{p}$ er et heltal
- $\boldsymbol{p} \geq 2$
- De eneste divisorer for $\boldsymbol{p}$ er 1 og $\boldsymbol{p}$

Eksempler:
- 2, 7, 19 er primtal
- −3, 1, 6 er ikke primtal

Primtalsopløsning af et positivt heltal $\boldsymbol{n}$:

$$\boldsymbol{n} = \boldsymbol{p}_1^{e_1} \times \boldsymbol{p}_2^{e_2} \times \ldots \times \boldsymbol{p}_k^{e_k}$$

Eksempel:
$200 = 2^3 \times 5^2$

Fundamental sætning i aritmetik:
Primtalsopløsningen af et positive heltal er unik

Største fælles divisor

Den **største fælles divisor** (GCD) for to positive heltal $\boldsymbol{a}$ og $\boldsymbol{b}$, betegnet gcd($\boldsymbol{a}$, $\boldsymbol{b}$), er det største positive heltal, der går op i både $\boldsymbol{a}$ og $\boldsymbol{b}$

Ovenstående definition kan udvides til vilkårlige heltal

Eksempler:
- gcd(18, 30) = 6 gcd(0, 20) = 20
- gcd(−21, 49) = 7

To heltal $\boldsymbol{a}$ og $\boldsymbol{b}$ siges at være **indbyrdes primiske**, hvis gcd($\boldsymbol{a}$, $\boldsymbol{b}$) = 1

Eksempel:
15 og 28 er indbyrdes primiske

Modulær aritmetik

Modulo-operatoren for et positivt heltal n

$r = a \bmod n$

er ækvivalent med

$a = r + kn$

og

$r = a - \lfloor a/n \rfloor\, n$

Eksempler:

$29 \bmod 13 = 3$	$13 \bmod 13 = 0$	$-1 \bmod 13 = 12$
$29 = 3 + 2 \cdot 13$	$13 = 0 + 1 \cdot 13$	$-1 = 12 + (-1) \cdot 13$

Modulo og GCD:

$\gcd(a, b) = \gcd(b, a \bmod b)$

Eksempel:

$\gcd(21, 12) = 3$

$\gcd(12, 21 \bmod 12) = \gcd(12, 9) = \gcd(9, 3) = \gcd(3, 0) = 3$

Euclids GCD-algoritme

300 f. Kr.

Euclids algoritme til beregning GCD benytter gentagne gange formlen

$\gcd(a, b) = \gcd(b, a \bmod b)$

Algorithm *EuclidGCD*(a, b)
Input integers a and b
Output $\gcd(a, b)$
if $b = 0$
  return a
return *EuclidGCD*$(b, a \bmod b)$

Eksempel:

$\gcd(412, 260) = 4$

a	412	260	152	108	44	20	4
b	260	152	108	44	20	4	0

Analyse

Algorithm *EuclidGCD*(a, b)
Input integers a and b
Output $\gcd(a, b)$
if $b = 0$
  return a
return *EuclidGCD*$(b, a \bmod b)$

Køretiden er proportional med antallet af rekursive kald af ***EuclidGCD***

Lad a_i og b_i være argumenterne til det i'te rekursive kald

Vi har

$a_{i+2} = b_{i+1} = a_i \bmod b_i = a_i \bmod a_{i+1} < a_{i+1}$

Følgen $a_1, a_2, \ldots, a_n$ aftager eksponentielt, idet vi kan vise, at

$a_{i+2} < 1/2\ a_i$ for $i > 1$

Tilfælde 1: $a_{i+1} \le 1/2\ a_i$: $a_{i+2} < a_{i+1} \le 1/2\ a_i$

Tilfælde 2: $a_{i+1} > 1/2\ a_i$: $a_{i+2} = a_i \bmod a_{i+1} \le a_i - a_{i+1} < 1/2\ a_i$

Derfor er det maksimale antal kald af algoritmen ***EuclidGCD***(a, b)

$1 + 2 \log \max(a, b)$

Algoritmen ***EuclidGCD***(a, b) udfører $O(\log \max(a, b))$ aritmetiske operationer

Multiplikativ invers (1)

Resterne modulo et positivt heltal n er mængden

$Z_n = \{0, 1, 2, \ldots, (n-1)\}$

Lad x og y være to positive elementer i Z_n, hvor

$xy \bmod n = 1$

Vi siger da, at y er **multiplikativ invers** til x i Z_n, og vi skriver $y = x^{-1}$

Eksempel:

Multiplikative inverse for resterne modulo 11

x	0	1	2	3	4	5	6	7	8	9	10
x^{-1}		1	6	4	3	9	2	8	7	5	10

Multiplikativ invers (2)

Sætning:
Et element $\boldsymbol{x}$ i $\boldsymbol{Z_n}$ har en multiplikativ invers, hvis og kun hvis $\boldsymbol{x}$ og $\boldsymbol{n}$ er indbyrdes primiske

Eksempel: Elementerne i $\boldsymbol{Z}_{10}$, der har en multiplikativ invers, er 1, 3, 5, 7

x	0	1	2	3	4	5	6	7	8	9
x^{-1}		1		7				3		9

Følgesætning:
Hvis $\boldsymbol{p}$ er et primtal, har alle rester i $\boldsymbol{Z_p}$, undtagen 0, en multiplikativ invers

Sætning:
Euclids GCD-algoritme kan udvides til at beregne den multiplikative inverse til et element $\boldsymbol{x}$ i $\boldsymbol{Z_n}$, eller afgøre, at en sådan ikke eksisterer

Potenser

Lad $\boldsymbol{p}$ være et primtal

Følgen af positive potenser af elementerne i $\boldsymbol{Z_p}$ indeholder gentagende delfølger

Længden af de gentagende delsekvenser og antallet af deres gentagelser er divisorerne til $\boldsymbol{p} - 1$

Eksempel ($\boldsymbol{p} = 7$): [$\boldsymbol{p} - 1 = 1{\cdot}6 = 2{\cdot}3$]

x	x^2	x^3	x^4	x^5	x^6
1	1	1	1	1	1
2	4	1	2	4	1
3	2	6	4	5	1
4	2	1	4	2	1
5	4	6	2	3	1
6	1	6	1	6	1

Fermats lille sætning


1601-65

Sætning:
Lad $\boldsymbol{p}$ være et primtal. For enhver rest $\boldsymbol{x} \neq 0$ i $\boldsymbol{Z_p}$ gælder, at $\boldsymbol{x}^{p-1}$ mod $\boldsymbol{p} = 1$

Eksempel ($\boldsymbol{p} = 5$):
1^4 mod 5 = 1
2^4 mod 5 = 16 mod 5 = 1
3^4 mod 5 = 81 mod 5 = 1
4^4 mod 5 = 256 mod 5 = 1

Følgesætning:
Lad $\boldsymbol{p}$ være et primtal.
For enhver rest $\boldsymbol{x} \neq 0$ i $\boldsymbol{Z_p}$ gælder, at den multiplikative inverse til $\boldsymbol{x}$ er $\boldsymbol{x}^{p-2}$ mod $\boldsymbol{p}$

Bevis:
$\boldsymbol{x}(\boldsymbol{x}^{p-2} \bmod \boldsymbol{p}) \bmod \boldsymbol{p} = \boldsymbol{x}\boldsymbol{x}^{p-2} \bmod \boldsymbol{p} = \boldsymbol{x}^{p-1} \bmod \boldsymbol{p} = 1$

Eulers sætning


1707-83

Den **multiplikative gruppe** for $\boldsymbol{Z_n}$, betegnet med $\boldsymbol{Z^*_n}$, er den delmængde af $\boldsymbol{Z_n}$, der er indbyrdes primiske med $\boldsymbol{n}$

Euler's **totientfunktion** af $\boldsymbol{n}$, betegnet med $\phi(\boldsymbol{n})$, er antallet af elementer i $\boldsymbol{Z^*_n}$

Eksempel:
$\boldsymbol{Z^*}_{10} = \{ 1, 3, 7, 9 \}$ $\qquad \phi(10) = 4$

Hvis $\boldsymbol{p}$ er et primtal, har vi
$\boldsymbol{Z^*_p} = \{1, 2, \ldots, (\boldsymbol{p}-1)\}$ $\qquad \phi(\boldsymbol{p}) = \boldsymbol{p} - 1$

Sætning:
For ethvert element $\boldsymbol{x}$ i $\boldsymbol{Z^*_n}$ gælder
$\boldsymbol{x}^{\phi(n)}$ mod $\boldsymbol{n} = 1$

Eksempel ($\boldsymbol{n} = 10$):
$3^{\phi(10)}$ mod 10 = 3^4 mod 10 = 81 mod 10 = 1
$7^{\phi(10)}$ mod 10 = 7^4 mod 10 = 2401 mod 10 = 1
$9^{\phi(10)}$ mod 10 = 9^4 mod 10 = 6561 mod 10 = 1

Eulers sætning er en generalisering af Fermats lille sætning

$\boxed{x^{-1} \equiv \boldsymbol{x}^{\phi(n)-1} \bmod \boldsymbol{n}}$

Kryptosystemet RSA

Kryptosystemet RSA

Opstilling:
- $n = pq$, hvor p og q er primtal
- e er indbyrdes primisk med $\phi(n) = (p-1)(q-1)$
- d er multiplikativ invers til e i $Z_{\phi(n)}$

Nøgler:
- Offentlig nøgle: $K_E = (n, e)$
- Privat nøgle: $K_D = d$

Kryptering:
- Klartekst M i Z_n
- Kodetekst $C = M^e \bmod n$

Afkryptering:
- $M = C^d \bmod n$

Eksempel:

Opstilling:
- $p = 7$, $q = 17$
- $n = 7 \cdot 17 = 119$
- $\phi(n) = 6 \cdot 16 = 96$
- $e = 5$
- $d = 77$ $\quad (77 \cdot 5 \bmod 96 = 1)$

Nøgler:
- Offentlig nøgle: $(119, 5)$
- Privat nøgle: 77

Kryptering:
- $M = 19$
- $C = 19^5 \bmod 119 = 66$

Afkryptering:
- $M = 66^{77} \bmod 119 = 19$

Komplet RSA eksempel

Opstilling:
- $p = 5$, $q = 11$
- $n = 5 \cdot 11 = 55$
- $\phi(n) = 4 \cdot 10 = 40$
- $e = 3$
- $d = 27$ $\quad (3 \cdot 27 = 81 = 2 \cdot 40 + 1)$

Kryptering:
- $C = M^3 \bmod 55$

Afkryptering:
- $M = C^{27} \bmod 55$

M	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
C	1	8	27	9	15	51	13	17	14	10	11	23	52	49	20	26	18	2
M	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
C	39	25	21	33	12	19	5	31	48	7	24	50	36	43	22	34	30	16
M	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54
C	53	37	29	35	6	3	32	44	45	41	38	42	4	40	46	28	47	54

Sikkerhed

Sikkerheden af RSA-kryptosystemet er baseret på den almindelige antagelse om, at faktorisering af store tal er beregningsmæssigt vanskelig

Den bedste kendte algoritme bruger tid, der er eksponentiel i antallet af bits i det tal, der skal faktoriseres

The RSA Challenge, der er sponsoreret af RSA Security, tilbyder pengepræmier for faktorisering af givne store tal

I april 2002 varierede præmierne fra 10,000\$ (576 bit) til 200,000\$ (2048 bit)

I 1999 blev et 512-cifret tal faktoriseret på 4 måneder ved brug af følgende computere:

160	175-400 MHz SGI og Sun
8	250 MHz SGI Origin
120	300-450 MHz Pentium II
4	500 MHz Digital/Compaq

Skønnet ressourceforbrug til faktorisering af et tal inden for 1 år:

Bit	PCer	Lager
430	1	128MB
760	215,000	4GB
1,020	342×10^6	170GB
1,620	1.6×10^{15}	120TB

Korrekthed

Vi viser korrektheden for RSA i det tilfælde, at $\boldsymbol{n}$ ikke går op i klarteksten $\boldsymbol{M}$

Vi viser, at der i dette tilfælde gælder, at
$(M^e)^d \bmod n = M$

Da $ed \bmod \phi(n) = 1$, er der et heltal k, for hvilket det gælder, at
$ed = k\phi(n) + 1$

Da n ikke går op i M, får vi ved hjælp af Eulers sætning, at
$M^{\phi(n)} \bmod n = 1$

Vi får således

$(M^e)^d \bmod n =$
$M^{ed} \bmod n =$
$M^{k\phi(n)+1} \bmod n =$
$MM^{k\phi(n)} \bmod n =$
$M\,(M^{\phi(n)})^k \bmod n =$
$M\,(M^{\phi(n)} \bmod n)^k \bmod n =$
$M\,(1)^k \bmod n =$
$M \bmod n =$
M

Beviset for korrekthed i det tilfælde, hvor $\boldsymbol{n}$ går op i klarteksten $\boldsymbol{M}$, kan ses i lærebogen

Algoritmiske problemer

Implementering af kryptosystemet RSA kræver forskellige algoritmer

- **Generelt**
Repræsentation af heltal af vilkårlig længde og aritmetiske operationer på disse

- **Kryptering**
Modulær potensopløftning

- **Afkryptering**
Modulær potensopløftning

- **Opstilling**
Generering af tilfældige tal med et givet antal bit (for at generere kandidater for $\boldsymbol{p}$ og $\boldsymbol{q}$)

Primtalitetstest (for at påvise, at kandidater for $\boldsymbol{p}$ og $\boldsymbol{q}$ er primtal)

Beregning af GCD (for at påvise, at $\boldsymbol{e}$ og $\phi(\boldsymbol{n})$ er indbyrdes primiske)

Beregning af den multiplikative inverse (for at beregne $\boldsymbol{d}$ ud fra $\boldsymbol{e}$)

Modulær potensopløftning

Brug af gentagen kvadrering øger beregningshastigheden for en modulær potens $a^p \bmod n$

Skriv eksponenten p på binær form:
$p = p_{b-1} p_{b-2} \cdots p_1 p_0$

Start med
$Q_1 = a^{p_{b-1}} \bmod n$
Beregn for $i = 2, 3, \ldots, b$
$Q_i = ((Q_{i-1})^2 \bmod n) a^{p_{b-i}} \bmod n$
Vi har, at
$Q_b = a^p \bmod n$

Denne algoritme udfører $O(\log p)$ aritmetiske operationer

Eksempel:
$3^{18} \bmod 19$ $(18 = 2^4 + 2^1 = 0x10010)$
$Q_1 = 3^1 \bmod 19 = \mathbf{3}$
$Q_2 = (\mathbf{3}^2 \bmod 19)3^0 \bmod 19 = \mathbf{9}$
$Q_3 = (\mathbf{9}^2 \bmod 19)3^0 \bmod 19 =$
$81 \bmod 19 = \mathbf{5}$
$Q_4 = (\mathbf{5}^2 \bmod 19)3^1 \bmod 19 =$
$(25 \bmod 19)3 \bmod 19 =$
$18 \bmod 19 = \mathbf{18}$
$Q_5 = (\mathbf{18}^2 \bmod 19)3^0 \bmod 19 =$
$(324 \bmod 19) \bmod 19 =$
$(17 \cdot 19 + 1) \bmod 19 = \mathbf{1}$

p_{5-i}	1	0	0	1	0
$3^{p_{5-i}}$	3	1	1	3	1
Q_i	3	9	5	18	1

Modulær potensopløftning i Java
(rekursiv udgave)

Hvis p er lige, er $a^p = (a \cdot a)^{p/2}$
Hvis p er ulige, er $a^p = a \cdot a^{p-1}$

```
long power(long a, long p, long n) {
    if (p == 0)
        return 1;
    long value = power((a * a) % n, p / 2, n);
    if (p % 2 == 1)
        value = (a * value) % n;
    return value;
}
```

Modulær potensopløftning i Java

(iterativ udgave)

```
long power(long a, long p, long n) {
    long value = 1;
    while (p > 0) {
        if (p % 2 == 1)
            value = (value * a) % n;
        a = (a * a) % n;
        p /= 2;
    }
    return value;
}
```


Modulær invers

Sætning:

Lad der være give to positive heltal, $\boldsymbol{a}$ og $\boldsymbol{b}$, og lad $\boldsymbol{d}$ være det mindste positive heltal, således at

$$d = ia + jb$$

for heltal $\boldsymbol{i}$ og $\boldsymbol{j}$.

Der gælder da, at

$$d = \gcd(a,b)$$

Eksempel:

$a = 21$

$b = 15$

$d = 3$

$i = 3, j = -4$

$3 = 3 \cdot 21 + (-4) \cdot 15 =$

$63 - 60 = 3$

Givet to positive heltal, $\boldsymbol{a}$ og $\boldsymbol{b}$, beregner den *udvidede* Euclids algoritme et trippel $(\boldsymbol{d,i,j})$, hvor

$d = \gcd(a,b)$

$d = ia + jb$

For at påvise eksistensen af, samt beregne den multiplikative inverse til $x \in Z_n$, udføres den udvidede Euclids algoritme på inputparret $(\boldsymbol{x,n})$

Lad $(\boldsymbol{d,i,j})$ være det trippel, der blev returneret

$d = ix + jn$

Tilfælde 1: $d = 1$

$\boldsymbol{i}$ er den inverse til $\boldsymbol{x}$ i Z_n

Tilfælde 2: $d > 1$

$\boldsymbol{x}$ har intet inverst element i Z_n

Den udvidede Euclids algoritme

Algorithm *ExtendedEuclidGCD*$(\boldsymbol{a}, \boldsymbol{b})$

Input Nonnegative integers $\boldsymbol{a}$ and $\boldsymbol{b}$

Output Triplet of integers (d, i, j) such that

$d = \gcd(a,b) = ia + jb$

if $b = 0$

return $(a, 1, 0)$

$(d, i, j) \leftarrow$ *ExtendedEuclidGCD*$(b, a \bmod b)$

return $(d, j, i - j\lfloor a/b \rfloor)$

a	412	260	152	108	44	20	4
b	260	152	108	44	20	4	0
a/b	1	1	1	2	2	5	
i	12	-7	5	-2	1	0	1
j	-19	12	-7	5	-2	1	0

Bevis ved induktion:

Basistilfældet: $b = 0$:

$a = \gcd(a, 0) = 1a + 0b$

Induktionshypotesen:

Antag, at

$(d, i, j) =$ *ExtendedEuclid*$(b, a \bmod b)$

er korrekt.

Vi har da, at

$d = \gcd(b, a \bmod b) = ib + j(a \bmod b) =$

$ib + j(a - \lfloor a/b \rfloor b) =$

$ja + (i - j\lfloor a/b \rfloor)b$

Da desuden $\gcd(b, a \bmod b) = \gcd(a, b)$,

må *ExtendedEuclid*(a, b) være korrekt

Pseudoprimalitetstest

- Antallet af primtal mindre end eller lig med $\boldsymbol{n}$ er cirka $n / \ln n$
- Afgørelse af om et tal er et primtal eller ej antages at være et "hårdt" problem
- Et heltal $n \geq 2$ siges at være et **base-*x* pseudoprimtal**, hvis
  $x^{n-1} \bmod n = 1$ (Fermats lille sætning)
- Sammensatte base-$\boldsymbol{x}$ pseudoprimtal er sjældne:
  Et tilfældigt 100-bit heltal er et sammensat base-2 pseudoprimtal med en sandsynlighed, der er mindre end 10^{-13}
  Det mindste sammensatte base-2 pseudoprimtal er 341
- Base-$\boldsymbol{x}$ pseudoprimalitetstest for et heltal $\boldsymbol{n}$:
  Afgør om $x^{n-1} \bmod n = 1$ (kan udføres effektivt med gentagen kvadrering)

Randomiseret primalitetstest

Vidnefunktion for sammensathed, ***witness***(x, n), med fejlsandsynlighed q for et heltal x:

Tilfælde 1: n er et primtal

$witness(x, n) = false$ (altid)

Tilfælde 2: n er sammensat

$witness(x, n) = false$ med sandsynlighed $q < 1$

Algoritmen ***RandPrimeTest*** afgør, om n er et primtal ved gentagne gange at evaluere ***witness***(x, n)

En variant af base-x pseudoprimalitet giver en god vidnefunktion for sammensathed (Rabin-Miller's algoritme)

Algorithm ***RandPrimeTest***(n, k)
Input integer n, confidence parameter k and composite witness function ***witness***(x,n) with error probability q
Output an indication of whether n is composite or prime with error probability 2^{-k}

$t \leftarrow k/\log_2(1/q)$ $\{q^t = 2^{-k}\}$
for $i \leftarrow 1$ **to** t
  $x \leftarrow random()$
  if $witness(x,n) = true$
    return "n is composite"
return "n is prime"

```
import java.math.BigInteger;
import java.util.Random;
import java.io.*;

public class RSA {
    public static void main(String[] args) throws IOException {
        int bitLength = 1024;
        Random rnd = new Random();
        BigInteger p = BigInteger.probablePrime(bitLength, rnd);
        BigInteger q = BigInteger.probablePrime(bitLength, rnd);
        BigInteger n = p.multiply(q);
        BigInteger phin = p.subtract(BigInteger.ONE).multiply(
                              q.subtract(BigInteger.ONE));
        BigInteger e;
        do
            e = new BigInteger(2 * bitLength, rnd);
        while (e.compareTo(phin) >= 0 || !e.gcd(phin).equals(BigInteger.ONE));
        BigInteger d = e.modInverse(phin);

        BufferedReader in = new BufferedReader(
                                  new InputStreamReader(System.in));
        while (true) {
            BigInteger ciphertext =
                new BigInteger(in.readLine().getBytes()).modPow(e, n);
            String plaintext =
                new String(ciphertext.modPow(d, n).toByteArray());
            System.out.println(plaintext);
        }
    }
}
```


Informationssikkerhed

Digital underskrift

En **digital underskrift** er streng S forbundet med en meddelelse M og dens forfatter A, som har følgende egenskaber:

- *Uafviselighed*: S identificerer entydigt forfatteren A af M og beviser, at A faktisk underskrev M
- *Integritet*: S godtgør, at M ikke er blevet ændret

Et skema for digital underskrift giver en algoritme til
(1) underskrivning af en meddelelse (foretages af forfatteren)
(2) verifikation af underskriften (foretages af læseren)

Underskrift: Alice (forfatteren) bestemmer $S = decrypt(K_D, M)$ ved brug af sin private nøgle K_D og sender parret (M,S) til Bob

Verifikation: Bob (læseren) bestemmer $M' = encrypt(K_E, S)$ ved brug af Alice's offentlige nøgle K_E og checker, at $M' = M$

Digital underskrift med RSA

Opstilling:

$n = pq$, hvor p og q er primtal

e er indbyrdes primisk med $\phi(n) = (p-1)(q-1)$

d inverse af e i $Z_{\phi(n)}$

Nøgler:

Offentlig nøgle: $K_E = (n, e)$

Privat nøgle: $K_D = d$

Underskrift:

Klartekst M i Z_n

Underskrift $S = M^d \bmod n$

Verifikation:

Check, at $M = S^e \bmod n$

Opstilling:

$p = 5$, $q = 11$

$n = 5 \cdot 11 = 55$

$\phi(n) = 4 \cdot 10 = 40$

$e = 3$

$d = 27$ $(3 \cdot 27 = 81 = 2 \cdot 40 + 1)$

Nøgler:

Offentlig nøgle: $K_E = (55, 3)$

Privat nøgle: $K_D = 27$

Underskrift:

$M = 51$

$S = 51^{27} \bmod 55 = 6$

Verifikation:

$M = 6^3 \bmod 55 = 216 \bmod 55 = 51$

Envejs-hashfunktion

En **envejs-hashfunktion** er en funktion H med følgende egenskaber:

- H afbilder en streng M af vilkårlig længde til et heltal $f = H(M)$ med et fast antal bit, kaldet **fingeraftrykket** af M
- H kan beregnes effektivt
- Givet et heltal f er det beregningsmæssigt vanskeligt at finde en streng M, således at $H(M) = f$
- Givet en streng M er det beregningsmæssigt vanskeligt at finde en anden streng M', således at $H(M) = H(M')$ (*kollisionsresistent*)
- Det er beregningsmæssigt vanskeligt at finde to strenge, M og M', således at $H(M) = H(M')$ (*stærkt kollisionsresistent*)

To udbredte anvendte envejs-hashfunktioner:
MD5 (Message Digest 5, 1992), som bruger et 128-bit (16 byte) fingeraftryk
SHA-1 (Secure Hash Algorithm 1, 1995), som bruger et 160-bit (20 byte) fingeraftryk

Fingeraftryk til digital underskrift

I RSA's skema til digital underskrift med modulo n skal meddelelsen, der skal underskrives, være et heltal i Z_n, d.v.s. meddelelsen kan højst være på log n bit

For at overvinde denne begrænsning kan vi bruge et **fingeraftryk** $f = H(M)$ af meddelelsen i stedet for meddelelsen selv, hvor H er en envejs-hashfunktion

Alice beregner først $f = H(M)$ og dernæst underskriften S af f
Bob beregner først $f = H(M)$ og verificerer derefter S

Hvis H er kollisionsresistent, er det beregningsmæssigt vanskeligt at ændre meddelelsen og samtidigt bevare underskriften for fingeraftrykket $f = H(M)$

meddelelse M → hash envejs → fingeraftryk $f = H(M)$ → underskriv → underskrift $S = f^d \bmod n$

Møntkast over nettet

Alice og Bob vil kaste en mønt tilfældigt i en kommunikation over Internettet

Følgende protokol baseret på en envejs-hashfunktion H sikrer imod snyd:

(1) Alice vælger et tilfældigt heltal x, beregner fingeraftrykket $f = H(x)$ og sender f til Bob
(2) Bob sender Alice sit gæt af, om x er lige eller ulige
(3) Alice bekendtgør resultatet af møntkastet: "ja", hvis Bob har gættet rigtigt; ellers "nej". Hun sender også x til Bob
(4) Bob verificerer, at Alice ikke har snydt, d.v.s. at $f = H(x)$

Hvis H er stærkt kollisionsresistent, er det beregningsmæssigt vanskeligt for Alice at snyde

Certifikater

Offentlig-nøgle kryptografi er baseret på enhver deltagers viden om de offentlige nøgler for de andre deltagere

Det er kompliceret at distribuere de offentlige nøgler til alle deltagere på en sikker måde

Et **certifikat** er en meddelelse af typen (*navn*, *offentlig nøgle*), der er underskrevet af en tredje part

En person eller virksomhed, som alle stoler på, en såkaldt **certificerings-autoritet** (CA), udsteder til hver deltager et certifikat (***Name***, $\boldsymbol{K_E}$), der autoritativt binder deltagerne til deres offentlige nøgler

Kun CA's offentlige nøgler behøver at blive distribueret

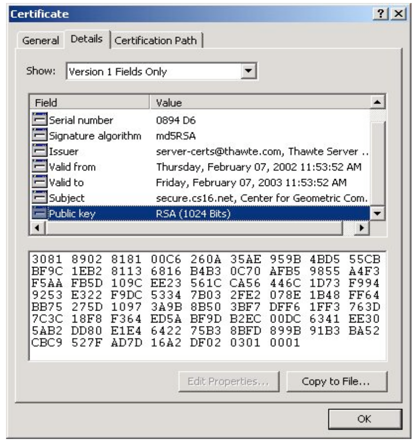
Inden en krypteret meddelelse sendes til Bob, eller en meddelelse underskrevet af Bob verificeres, bestemmer Alice Bob's offentlige nøgle ved hjælp af Bob's certifikat, (Bob, $\boldsymbol{K_E}$)

Web server certifikater

Et **web server certifikat** bruges til at autentificere den offentlige nøgle for en web server

Felter i et web server certifikat
- Serienummer
- Hash- og underskriftskema (f.eks. MD5 og RSA)
- Udgiver (certificeringsautoritet)
- Gyldighedsperiode (fra, til)
- Subjekt (URL og organisation)
- Offentlig nøgle

Protokollen SSL (Secure Socket Layer) bruger web server certificering for at tilbyde kryptering og autentificering ved en sikker web forbindelse (http)



Ophør af certifikater

Under visse omstændigheder er det nødvendigt at ophæve et certifikat inden dets udløbsdato
- Den private nøgle er blevet afsløret
- Certifikatet blev fejlagtigt udgivet af CA

Certifikatophørsliste
- Tidsstemplet liste over alle ikke-udløbne certifikater, der er blevet ophævet af CA
- Publiceres periodevis og underskrives af CA

Når man præsenteres for et certifikat, skal man
(1) Verificere CA's underskrift på certifikatet
(2) Checke at certifikatet ikke er blevet ophævet ved at søge i den senest tilgængelige certifikatophørsliste

Web-browsere checker normalt ikke status for en web servers certifikat, hvilket udgør en sikkerhedsrisiko