LaborationsPM
Grupp 1
Kryptering smetoder och säkring av datasystem
Kursansvarig: Eric Järpe
Högskolan i Halmstad

Laboration 2: Kryptering med publika nycklar

I ett krypteringssystem skriver (krypterar) avsändaren sitt meddelande på ett sådant sätt att endast behöriga mottagare kan rekonstruera (dekryptera) det ursprungliga meddelandet (klartexten). Innan meddelandet krypteras omvandlas det till ett positivt heltal M. Man kan t.ex. använda $A = 1$, $B = 2$, $C = 3$, o.s.v. eller ASCII-koden för respektive tecken för att få klartextmeddelandet på denna sifferform.

Ett krypteringssystem kan sägas bestå av två funktioner: K som krypterar och D som dekrypterar, så att

$$D(K(M)) \;=\; M \quad \text{för alla tal } M$$

d.v.s. om man dekrypterar ett krypterat meddelande så får man tillbaka meddelandet i klartextform. Funktionerna K och D innehåller i de flesta krypteringssystem parametrar som kallas krypteringsnycklar respektive dekrypteringsnycklar.

Ett av de idag mest kända, spridda och på många sätt bästa krypteringssystemen är RSA-kryptot från 1977 uppkallat efter sina upphovsmän Ronald L. Rivest, Adi Shamir och Leonard M. Adleman och det anses erbjuda mycket hög säkerhet. RSA är patentskyddat i USA men får användas fritt av privatpersoner. Många av de stora mjukvaruföretagen som t ex Microsoft, IBM, Adobe och Apple använder idag systemet. Krypteringsprogrammet PGP (Pretty Good Privacy), som bl a används för säker email-korrespondens, är ett annat exempel som bygger på RSA.

Ett annat krypteringssystem som nog kan anses vara det näst vanligaste public-key-systemet är ElGamal. Detta är inte lika förbluffande enkelt som RSA men relativt de andra metoder som finns definitivt ett av de enklare. Det bär sitt namn efter uppfinnaren Tahere ElGamal, en egyptisk kryptograf som 1985 introducerade systemet i en vetenskaplig artikel. Det är speciellt på så vis att det, till skillnad från RSA, är ett exempel på probabilistisk kryptering. Detta innebär att avsändaren slumpmässigt kan välja olika privata nycklar (vilket verkligen resulterar i olika kryptotext) men som oavsett vilken dekrypteras av mottagaren med samma dekrypteringsnyckel till samma klartext.

Vi ska använda båda dessa krypteringssystem i denna laboration, först RSA och sedan ElGamal. För att förklara själva krypteringsalgoritmerna behöver vi känna till några grundläggande resultat.

Definition 1 *Om a, b är heltal sådana att $\gcd(a, b) = 1$ så kallas a och b* **relativt prima**.

Definition 2 *För positiva heltal n är* **Eulers ϕ-funktion** *$\phi(n)$ det antal positiva heltal mindre än n som är relativt prima med n.*

Observera att om p är ett primtal så är $\phi(p) = p - 1$.

Sats 1 **Eulers produktformel**
Om $\prod_{i=1}^{k} p_i^{m_i}$ är primtalsfaktoriseringen av talet n är $\phi(n) = \prod_{i=1}^{k} p_i^{m_i - 1}(p_i - 1)$

Observera att om p och q är primtal så är pq primtalsfaktoriseringen av $n = pq$. Därmed är $\phi(n) = (p - 1)(q - 1)$ i detta fall.

Sats 2 Eulers sats
Om a och n är heltal, $n \geq 2$ och $\gcd(a,n) = 1$, så är $a^{\phi(n)} \equiv 1 \pmod{n}$.

Exempel Antag att vi har ett tal n och att vi väljer två tal e och d så att $ed \equiv 1 \pmod{\phi(n)}$. Vi ska nu visa att $M^{ed} \equiv M \pmod{n}$.

Villkoret att $ed \equiv 1 \pmod{\phi(n)}$ innebär att ed ger resten 1 vid heltalsdivision med $\phi(n)$, d.v.s. (enligt divisionsalgoritmen) att det finns ett tal k sådant att $ed = k\phi(n)+1$. Dessutom är (enligt Eulers sats) $M^{\phi(n)} \equiv 1 \pmod{n}$. Därmed är

$$\begin{aligned} M^{ed} &= M^{k\phi(n)+1} \\ &= M^{k\phi(n)} \cdot M^1 \\ &= (M^{\phi(n)})^k \cdot M \\ &\equiv 1^k \cdot M \\ &= M \pmod{n} \end{aligned}$$

□

I båda krypteringssystemen behöver vi kunna beräkna diskreta inversen till ett tal a mod n och detta görs genom att beräkna $\gcd(a,n)$. Under förutsättning att denna är 1 kan sedan beräkningen nystas upp baklänges och diskreta inversen bestämmas. Om emellertid $\gcd(a,n) \neq 1$ så existerar inte denna diskreta invers.

Definition 3 *En generator g av ett ändligt fält $\mathbb{F}_n$ är ett tal $g \in \mathbb{F}_n$ sådant att $\{g^k : k \in \mathbb{F}_n\} = \mathbb{F}_n$.*

Exempel Betrakta fältet $\mathbb{F}_7 = \{1,2,3,4,5,6\}$. Här gäller att $2^1 = 2$, $2^2 = 4$, $2^3 = 8 \equiv 1 \pmod{7}$ varmed $2^4 \equiv 2$ och allt börjar om igen. Därmed är $\{2^k : k \in \mathbb{F}_7\} \subset \mathbb{F}_7$, d.v.s. talet 2 är inte en generator av $\mathbb{F}_7$. Däremot är $3^1 = 3$, $3^2 \equiv 2$, $3^3 \equiv 3 \cdot 2 = 6$, $3^4 \equiv 3 \cdot 6 \equiv 4$, $3^5 \equiv 3{\cdot}4 \equiv 5$ och $3^6 \equiv 3{\cdot}5 \equiv 1$ d.v.s. $\{3^k : k \in \mathbb{F}_7\} = \mathbb{F}_7$, d.v.s. talet 3 är en generator av $\mathbb{F}_7$. □

Vid kryptering med ElGamal behöver vi kunna välja en gärna stor generator av för ett fält av hög primtalsordning p. Då blir det praktiskt omöjligt med ett program som testar fram en generator med det resonemang som användes i exemplet ovan. Det är istället oerhört användbart med följande resultat.

Sats 3 Generatorsatsen
Om talet $g \in \mathbb{F}_{2p+1}$ uppfyller villkoren $g^2 \not\equiv 1 \pmod{2p+1}$, $g^p \not\equiv 1 \pmod{2p+1}$ där p är ett primtal sådant att $2p+1$ också är det, så är g en generator av fältet $\mathbb{F}_{2p+1}$.

I RSA-systemet har varje deltagare både publika och privata krypteringsnycklar och det är viktigt att skilja på vilka som är publika och vilka som är privata! I förklaringen nedan är nycklarna p, q, $\phi(n)$, d privata och e, n publika.

Antag nu att Alice vill skicka ett hemligt meddelande till Bob. Först och främst väljer båda privat respektive publika nycklar. Om vi börjar med att Alice meddelande till Bob måste Bob ha valt primtal p_B och q_B och en krypteringsnyckel e_B. Talen p och q ska vara så stora att de har fler siffror än det meddelande (d.v.s. tal) som ska krypteras. Beträffande e_B ska detta tal vara relativt prima med $\phi(p_Bq_B) = (p_B - 1)(q_B - 1)$. Detta är mycket viktigt för annars existerar ingen dekrypteringsnyckel! Då ett sådant e_B valts (gärna så litet som möjligt så att krypteringsalgoritmen blir enkel och snabb) publicerar Bob $n_B = p_Bq_B$ och e_B medan han kan beräkna dekrypteringsnyckeln $d_B = e_B^{-1} \bmod \phi(n_B)$. Bobs hemliga nycklar

är alltså p_B, q_B och d_B medan hans publika är n_b och e_B.

Metoder som är snabba i Python och andra miljöer

Värt att notera är att i Python har man stor hjälp av att kunna beräkna diskret exponentiering a^b mod c för väldigt stora tal. Även om den gamla klassikern `(a**b)%c` funkar förvånansvärt bra för stora `a` och `c` blir problemen större när dessutom `b` är stort. (Stort i detta sammanhang kan vara 50-siffriga tal eller större.) Då funkar däremot `pow(a,b,c)` desto bättre.

För att lösa t.ex. den diofantiska ekvationen $ax + by = c$ kan man använda `diop_solve(a*x + b*y - c)`. Denna returnerar en tupel av strängar där strängarna anger en formel för beräkning av alla lösningar till ekvationen. För att använda funktionen `diop_solve` måste den importeras från biblioteket `sympy.solvers.diophantine`. Dessutom måste `x,y` importeras från `sympy.abc`.

För primtalsbestämning används lämpligen `isprime` och för beräkning av gcd används `gcd`. Slumpmässiga primtal kan genereras med `randprime`. Alla dessa importeras från biblioteket `sympy`, de funkar för stora tal och är mycket snabba. (Generering av slumpmässiga primtal kan även ännu snabbare göras på `https://bigprimes.org` men då måste genereringen göras utanför Python.)

När det gäller generering av slumpmässiga heltal funkar det dock fint med gamla hederliga `randint` och för slumpmässig permutering av elementen i en lista gör `shuffle` ett förträffligt arbete, precis som vanligt. Båda dessa importeras så klart från `random`.

Primtalsfaktorisering görs snabbast i Python med metoden `primefac` från biblioteket `primefac`. För att få reda på primtalsfaktoriseringen måste dock även `list` appliceras på resultatet. När de tal som ska faktoriseras är uppåt 60 siffror och mer rekommenderas dock faktoriseraren av Dario Alpertron som är publikt tillgänglig från sidan `https://www.alpertron.com.ar/ECM.HTM`. Den klarar att faktorisera tal med upp till 300 siffror.

Ett fall när Python kommer till korta är om man t.ex. ska beräkna vissa rötter ur jämna potenser. Antag att man beräknar tredje roten ur 60-siffriga kuber.

```
m = randint(10**19,10**20-1)
n = n**k
print(m,int(n**(1/k)))
```

Med t.ex. `k=2` och `m=78369452225718919394` fås `int(n**(1/k))=78369452225718910976` (d.v.s. avvikelse i sjuttonde siffran) och med `k=3` fås `int(n**(1/k))=78369452225718714368` d.v.s. en avvikelse i femtonde siffran. Med större tal blir felet naturligtvis större. Detta kan vara förödande när det gäller kryptering och kryptoanalys. Mathematica dock beräkna `n**(1/k)` exakt rätt även för mycket stora tal.

Kryptering med RSA

1. Avsändaren Alice transformerar sitt hemliga meddelandet i klartext till ett heltal M_A, enligt något känt system (t.ex. ASCII-kod).

2. Hon beräknar sedan
$$C_A = M_A^{e_B} \bmod n_B$$
där (e_B, n_B) är Bobs publika nyckel.

3. Skicka talet C_A (kryptot) till mottagaren Bob.

Viktigt i den praktiska användingen av RSA är alltså att Alice använder Bobs publika nycklar för att kryptera sitt meddelande. Och Bob använder Alices publika nycklar för sitt svar till Alice.

Alice har precis som Bob valt egna primtal p_A och q_A och en egen krypteringsnyckel e_A. På samma sätt som för Bobs nycklar ska Alices p_A och q_A ha så många siffror att produkten $n_A = p_a q_a$ har fler siffror än antalet siffror i det tal (Bobs svar M_B till Alice) som ska krypteras. Dessutom ska hennes publika nyckel e_A vara relativt prima med hennes hemliga $\phi(p_A q_A)$. Slutligen är hennes hemliga dekrypteringsnyckel $d_A = e_A^{-1} \bmod \phi(p_A q_A)$. Alices hemliga nycklar är p_A, q_A och d_A medan hennes publika är n_A och e_A.

Dekryptering med RSA

1. Mottagaren Bob tar emot det krypterade meddelandet, C_A.
2. Dekrypteringen gör Bob sedan genom att beräkna

$$C_A^{d_B} \bmod n_B$$

   Då är $C_A^{d_B} = (M_A^{e_B})^{d_B} \bmod n_B$ vilket vi beräknade i exemplet ovan till M_A.

Säkerheten i RSA-systemet hänger på att det för närvarande inte finns någon effektiv algoritm för att primtalsfaktorisera heltal. Väljer vi bara p och q tillräckligt stora (mer än 150 siffror) finns det ingen möjlighet att faktorisera (300-siffriga) $n = pq$ inom en rimlig tidsrymd. Som exempel kan nämnas att år 2009 behövde en modern PC fem månader för att primtalsfaktorisera ett 232-siffrigt tal (RSA-768 challenge list) med den bästa tillgängliga faktoriseringsalgoritmen, GNFS (General Number Field Sieve).

Kryptering med ElGamal

1. Mottagaren Bob har i förväg genererat nycklar: p (publik primtalsordning), b (privat), g (publik generator av $\mathbb{F}_p$) och $\beta = g^b \bmod p$ (publik).
2. Avsändaren Alice transformerar sitt hemliga meddelandet i klartext till ett heltal M_A, enligt något känt system (t.ex. ASCII-kod).
3. Hon väljer sedan $k_A \in \mathbb{F}_p$ (privat) och beräknar $r_A = g^{k_A} \bmod p$ och $t_A = M_A \beta^{k_A} \bmod p$.
4. Alice skickar "kryptotexten" (r_A, t_A) till Bob.

Dekryptering med ElGamal

1. Bob tar emot "kryptotexten" (r_A, t_A) från Alice.
2. Han dekrypterar genom att beräkna $M_A = t_A r_A^{-b} \bmod p$ där $r_A^{-b} \bmod p = (r_A^{-1} \bmod p)^b \bmod p$ och r_A^{-1} är den diskreta inversen till r_A i $\mathbb{F}_p$.

Säkerhetsprincipen för ElGamal-systemet är DLP, *Discrete Logarithm problem*, som innebär att en person som får reda på de publika nycklarna p, g och β ska beräkna b sådant att $g^b \bmod p = \beta$. Då kan nämligen den personen även dekryptera: $t_A r_A^{-b} \bmod p$ och därmed få reda på klartexten M_A. Varken RSA eller ElGamal är säkra mot framtida kvantdatorer med hög kapacitet.

Laborationsuppgifter

Räkna med Python men bara med operatorerna `+`, `-`, `*`, `/`, `//`, `**` och `%` (inga andra fördefinerade funktioner) i uppgifterna 1–5.

1. Beräkna $\gcd(33\,945, 36\,208)$.

2. Primtalsfaktorisera talet $2\,385\,040\,224$.

3. Beräkna

   (a) $1\,579^{4\,797\,011\,289\,608} \bmod 2\,385\,040\,224$ för hand m.h.a. Eulers sats. (Observera att ordningen är samma tal som i uppgift 2. Du kan beräkna $\phi(n)$ för lämpligt n med hjälp av Eulers produktformel. Kontrollera ditt svar med Mathematicafunktionerna `EulerPhi` och `PowerMod` eller med motsvarande funktioner i Python.)

   (b) $1\,579^{-4\,797\,011\,289\,608} \bmod 2\,385\,040\,224$ för hand m.h.a. resultatet i 3(a) och Euklides algoritm.

4. Bestäm några olika värden på krypteringsnyckeln e om $p = 599$ och $q = 617$. Välj därefter ett av värdena på e, bestäm motsvarande dekrypteringsnyckel d och kryptera meddelandet

   $$M = 181\,901$$

   Kontrollera dina beräkningar genom att dekryptera det krypterade meddelandet. Vilka tal M kan krypteras med ovanstående nycklar?

5. Verifiera att $g = 3$ är en generator av fältet $\mathbb{F}_{31}$.

6. Antag nu att Alice ska ElGamal-kryptera ett kort textmeddelande till Bob. Skriv de tre Pythonprogrammen `elgamal_keys` (som generarar publika och privata nycklar för meddelanden upp till en given längd), `elgamal_encr` (som krypterar ett meddelande med de publika nycklarna från `elgamal_keys`) och `elgamal_decr` (som dekrypterar meddelandet given kryptotexten och de publika och privata nycklarna från `elgamal_keys`) enligt instruktionerna nedan.

   (a) Skriv först

      i. en funktion `gen_p` som slumpmässigt väljer en primtalsordning `p` och en generator `g` av fältet $\mathbb{F}_p$. Funktionen ska klara att leverera en 60-siffrig generator `g` och en 200-siffrig primtalsordning `p` på tre minuter.

      ii. en funktion `choose_b` som väljer ett slumpmässigt fyrsiffrigt heltal `b`. (Observera att detta innebär att alla 4-siffriga tal ska vara lika sannolika vid valet av `b`.)

   Skriv sedan programmet `elgamal_keys` som använder funktionerna `gen_p` och `choose_b` för att tillhandahålla en tupel av nycklarna `(g,p,b,beta)` där `beta` är den publika nyckeln $\beta = g^b \bmod p$. Programmet ska klara att leverera dessa nycklar på mindre än tre minuter.

(b) Skriv sedan

i. funktionen `digitize` som tar en textsträngen, `s`, som input och returnerar heltalet `m` som output, där siffrorna i `m` i grupper om 3 motsvarar ASCII-värdet för respektive tecken ur strängen `s`. T.ex. ska strängen `s = 'HEJ'` resultera i talet `m = 72069074`. (Observera att den inledande nollan försvinner då talet $072 = 72$.)

Sedan ska krypteringsprogrammet `elgamal_encr` givet `p`, `g`, `beta` och klartexten `plain` i form av en sträng och returnerar kryptot `(rA,tA)` som är en tupel av heltal. Dessa heltal ska beräknas enligt definitionen av ElGamal-kryptering med snabba Pythonmetoder så att det funkar för stora värden på alla nycklar.

(c) Slutligen ska

i. funktionen `inv` skrivas. Denna ska, givet ett tal `a` och en ordning `n`, returnera den diskreta inversen $\texttt{a}^{-1}$ mod `n`. Den ska klara att leverera svar för 60-siffriga tal `a` och 200-siffrig ordning `n` på mindre än en sekund. Om inversen inte existerar ska den returnera värdet `0`.

ii. funktionen `letterize` skrivas. Den ska ta ett heltal `m` som input och returnera textsträngen `s` som output. Tecknen i strängen `s` ska motsvara de tecken som har ASCII-värdena i grupper om 3 siffror i talet `m`. T.ex. ska talet `m = 72069074` resultera i strängen `s = 'HEJ'`. (Observera att den inledande nollan försvinner då talet $072 = 72$.)

Allra sist ska krypteringsprogrammet `elgamal_decr` givet de publika nycklarna `p`, `g`, `beta`, kryptot `(tA,rA)` och den privata nyckeln `b` genomföra dekrypteringen enligt konstens regler och som output returnera klartexten i form av strängen `s`. Givetvis med snabbaste Pythonmetoder så att leverans av resultat sker inom rimlig tid.

(d) Demonstrera hela proceduren med nyckelgenerering, kryptering och dekryptering m.h.a. av programmen för klartexten HJÄRNAN UTI KNUTAR VRIDES NÄR JAG TÄNKER PÅ EUKLIDES.

För Godkänd lab ska minst 5 uppgifter vara helt korrekt lösta.

För 1 bonuspoäng ska alla uppgifter vara korrekt lösta.

LYCKA TILL!