

Kryptografins grunder: hental 2

Erik Tjernlund*

750519-0137

d98-etj

2 maj 2002

1 Primalitetstest

Uppgiften var att hitta ett primtal p så att $p = 2q + 1$ där q också är ett primtal. Valde att implementera Miller-Rabins algoritm (i Java) för att kontrollera stora tals primalitet.

Visserligen fick man använda Javas inbyggda `BigInteger` för att representera stora heltal, men bara de grundläggande räknesätten i den. Skrev därför en wrapper-klass (komposition istället för arv) till `BigInteger`, `MyBigInt`.

För att kunna utföra Miller-Rabin behövde jag en exponentieringsfunktion för stora tal som räknade i modulo. Använde algoritmen som återfinns i Stinson för detta.

Slutligen implementerade jag Miller-Rabin i `MyBigInt` och skrev ett program som använde klassen för att få fram några p genom att slumpa fram stora tal och sedan kontrollera om de var prima eller inte. Körtiden för att få fram ett p varierar, men omkring 20 – 30 sekunder verkar vara ganska normalt.

För att vara hyggligt säker på att få ett trovärdigt svar från Miller-Rabin körde jag den 100 gånger. Då blir sannolikheten för ett korrekt svar $(\frac{1}{4})^{100}$ och det borde vara tillräckligt.

$q = 6881995807579240721$

$p = 13763991615158481443$

$q = 7762673882241051461$

$p = 15525347764482102923$

$q = 11394770342837685773$

$p = 22789540685675371547$

Kontrollerade dessa p med hjälp av Maple och alla är primtal.

*I samarbete med Emanuel Viklund (d98-evi)

2 Kollisioner i försvagad SHA-1

Hann inte implementera denna klart, men eftersom jag vid första hemtalsredovisningen fick tipset att det kunde vara värt att skriva ner halvfärdigt arbete, så beskriver jag hur jag tänkte att jag skulle göra.

Uppgiften är att hitta en kollision i en försvagad variant av SHA-1 hashning (160 bitars block in och 50 bitars hashvärde ut). Den första tanken är en naiv uttömmande sökning. Stoppar man in 2^{50} olika indata, måste två av dessa mappas till samma uthash. På grund av födelsedagsparadoxen kan man en sannolikhet på $\frac{1}{2}$ få en kollision efter $\sqrt{2^{50}} = 2^{25}$ olika indata. Detta är dock fortfarande ganska mycket data och om man ska spara både indata och hashvärde för varje så blir det flera gigabyte. Inte så lyckat.

Om man använder Pollard-Rhos algoritm kan man upptäcka en kollision utan att behöva spara datat i internminnet. Problemet blir dock att när Pollard-Rho upptäcker en kollision, så kan man inte veta vilket indata som gav upphov till samma hashvärde (till skillnad från när man använder algoritmen för att hitta faktorer till stora sammansatta heltal, eftersom om man hittar en faktor överhuvudtaget, så är man klar).

Man får spara in- och utdata på en fil och sedan sortera och leta i denna fil när Pollard-Rho har signalerat att en kollision har inträffat.

Som sagt. Tyvärr hann jag inte implementera denna lösning.

3 RSA och faktorisering

Använde min `MyBigInt` från den första uppgiften. Skrev till en version av euklidisk algoritm (från Stinson) och en `sqrt()`-metod i den. Skrev ett litet java-program som implementerade Wieners algoritm och fann faktorerna:

```
p = 1674960498834085812920457916453747019461644031395307920624947349951043704631
q = 1473676470788342281338779191792495900393751209539300628363443011313652145299
```

4 Kollisioner i SHA-1

Uppgiften är att visa att om vi har en kollision i SHA-1, $H(x) = H(y)$ och $x \neq y$, så går det att konstruera en kollision i SHA-1s kompressionsfunktion.

Om x är lika lång som y vet vi att deras respektive H_0, H_1, H_2, H_3 och H_4 till en början kommer vara olika. Eftersom vi vet att $H(x) = H(y)$ måste det betyda att de någonstans på vägen får samma värde.

Undersöker man algoritmen ser man att kompressionssteget är det enda stället som H_0, H_1, H_2, H_3 och H_4 får nya värden. Kompressionsfunktionens indata är data som varit olika i tidigare steg. Detta betyder alltså att för att det ska bli samma uthash i slutändan, så måste kompressionen mappa två olika indata till samma utdata någonstans i kedjan.

Om x och y är olika långa inser man att det sista blocket som ska hashas kommer skilja sig från varandra (på grund av paddningen med meddelandets längd). Kompressionsfunktionen måste alltså generera samma utdata från två olika indata och därmed har vi en kollision även i detta fall.

5 ElGamalsignaturer

6 Exponent i RSA

Om N_A , N_B och N_C inte är parvis relativt prima, kan man beräkna till exempel $p = \gcd(N_A, N_B)$. Då har vi hittat en gemensam faktor och kan då enkelt dekryptera meddelandet x .

När N_A , N_B och N_C är relativt prima kan vi med hjälp av den kinesiska restsatsen hitta $y \in \mathbb{Z}_{N_A N_B N_C}^*$ sådant att $y \equiv x^3 \pmod{N_A, N_B, N_C}$. Det betyder att $y \equiv x^3 \pmod{N_A N_B N_C}$, men eftersom $x < \min(N_A, N_B, N_C)$ och därmed $x^3 < \min(N_A, N_B, N_C)^3 < N_A N_B N_C$ är $y = x^3$. Meddelandet x kan alltså beräknas med $x = \sqrt[3]{y}$.

7 Generatoralgoritm

Klarar tyvärr inte av hela uppgiften, men kan bevisa den ledtråd som ges i slutet. Alltså att visa att om och endast om $\gcd(p-1, r) = 1$ så är g^r en generator till \mathbb{Z}_p^* .

Sätter $a = |g^r|$ där $1 \leq a \leq p-1$ och då gäller att $g^{ra} = 1 \pmod{p}$. Eftersom $g^{p-1} = 1 \pmod{p}$ är ra en multipel av $p-1$.

Då $\gcd(p-1, r) = 1$ är a en multipel av $p-1$, men vi vet att $a \leq p-1$. Alltså är $a = p-1$ och g^r är en generator.

Nu ska vi visa att om $\gcd(p-1, r) = b$ där $1 \leq b \leq p-1$, så är g^r inte en generator till \mathbb{Z}_p^* .

Vi har $(g^r)^{\frac{p-1}{b}} \equiv (g^{p-1})^{\frac{r}{b}} = 1 \pmod{p}$ och därmed är g^r inte en generator till kroppen $\implies g^r$ är en generator till kroppen \mathbb{Z}_p^* om och endast om, $\gcd(p-1, r) = 1$.