Den elliptiske fælde..

Primtalsfaktorisering med LENSTRAs elliptiske kurve metode (ECM/ECF)

Af Christel Bach
Jan. 2006

Om denne artikel:

I artiklen "Factorization of the tenth fermat number" af Richard P.Brent gennemgåes Lenstras elliptiske kurve faktoriseringsmetode i flere faser indklusiv flere forbedringer, til at faktorisere det tiende fermattal.
I denne artikel vil jeg med baggrund i artiklen løse følgede opgave stillet som eksamensopgave i KryptologiB ultimo 2005 ved Århus Universitet:
Med baggrund i artiklen R. Brent "Factorization of the tenth Fermat number" ønskes en detaljeret beskrivelse af Lenstras elliptiske kurve faktoriseringsalgoritme (ECF) med de beskrevne forbedringer. Beskrivelsen skal skal lægge op til en selvstændig implementation af ECF som kan faktorisere tallet

4903985730770843887365515143614063711995422120485270325 9

Indledning

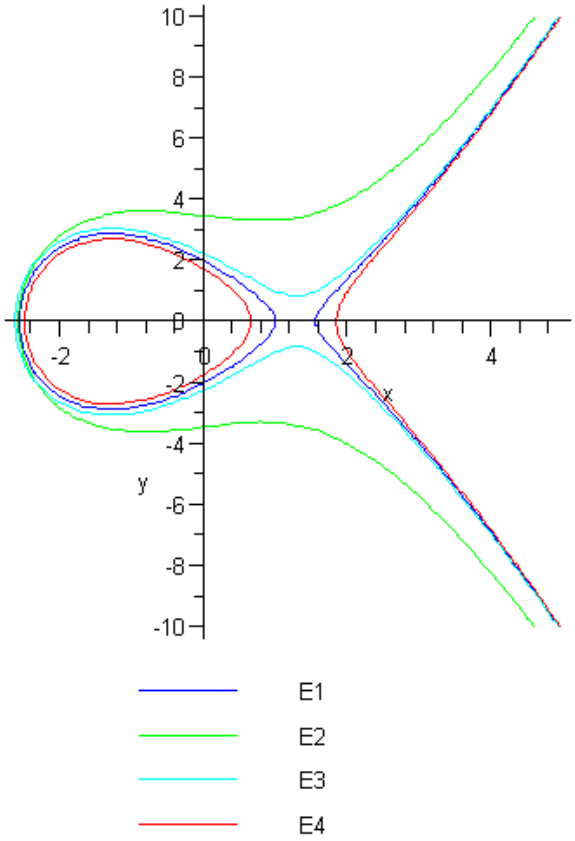
Pollards p-1 metode er en kendt og anvendt metode til primtalsfaktorisering. Imidlertid fik Lenstra i 1985 en ide til forbedring af denne metode ved at basere den på elliptiske kurver
I artikelen vil jeg derfor først se på Pollards p-1 metode. Lenstras algoritme bruger i stor udstrækning aritmetik på elliptiske kurver og jeg vil derfor kort vil ridse op hvad elliptiske kurver er og hvordan man regner på dem.
Dernæst vil jeg se på selve ECF/ECM algoritmen som den fremstilles i nævnte artikel og herefter ridse de nævnte forbedringer op. Jeg vil desuden se på andre kilders fremstilling af algoritmen
Jeg vil slutte af med en række implementationsovervejelser, der også ser på kendte implementationer og deres effektivitet.

Elliptiske kurver:

En elliptisk kurve kan antage denne såkaldt korte Weierstrass form:

$$y^2 = x^3 - Ax + B \tag{1.1}$$

Er kurverne defineret på de reelle tal får vi afbildninger som f.eks nedenstående:

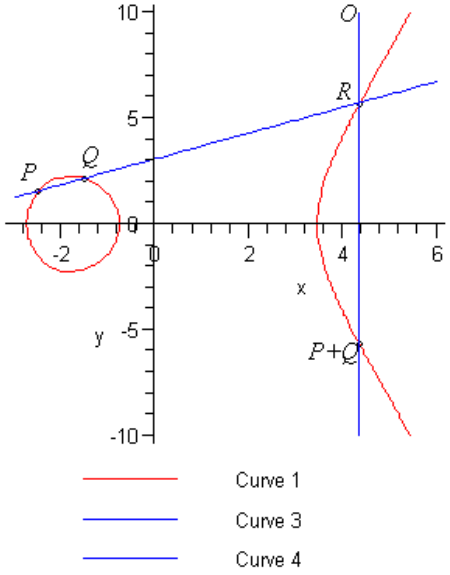


Kurven er genereret i Maple med følgende kommando:

```
> restart:with(plots):with(numtheory):with(algcurves):
> implicitplot([y^2=x^3-5*x+4,y^2=x^3-2*x+12,y^2=x^3-
5*x+5,y^2=x^3-5*x+3],x=-5..7,y=-
10..10,numpoints=6000,color=[blue,green,cyan,red],
legend=[E1,E2,E3,E4]);
```

Vi for langer dog at kurven skal være glat, dvs rødderne i kubikken skal være forskellige dvs diskriminanten d ikke må være nul, dvs $d = ((r_1 - r_2)(r_1 - r_3)(r_2 - r_3))^3 = -(4A^3 + 27B^2) \neq 0$

Addition af to punkter P og Q kan visualiseres således:

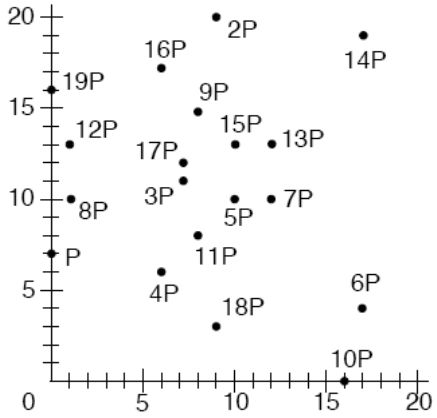


I grove træk er metoden altså følgende: Linien der forbinder de to punkter forlænges til den skærer kurven et tredie sted og denne spejles. Adderes i stedet P+P vælges tangentlinien i punktet P og denne forlænges ligeledes til den skærer kurven og reflekteres.
Anvender vi den korte Weierstrass form (1.1) vil punktet P+Q få koordinater der beregnes som følger:
P+Q:
$x_1 \neq x_2$ og $y_1 \neq y_2$, altså P er forskellig fra Q så har vi at:
(1.2) $x_3 = m^2 - x_1 - x_2$, $y_3 = m(x_1 - x_3) - y_1$ hvor m er hældningen: $m = y_2 - y_1 / x_2 - x_1$

Hvis P og Q er sammenfaldende, dog $y_1 \neq 0$:
2P:
(1.3) $x_3 = m^2 - 2x_1$, $y_3 = m(x_1 - x_3) - y_1$ hvor hældningen i stedet beregnes som tangenten i punktet
$m = 3x_1^2 + A / 2y_1$

Vi opererer ligeledes med et punkt i uendelig ∞, repræsenteret ved en lodret linie. Der gælder at P+Q=∞, hvis $x_1 = x_2$ men $y_1 \neq y_2$.
Der gælder ligeledes at P+∞,=P for alle P.
Punkterne på den elliptiske kurve opfylder betingelserne for en additiv abelsk gruppe med identitetselement ∞.
Gruppen består altså af mængden af punkter (x,y) der ligger i K x K som ligger på kurven samt et punkt i uendelig, som jeg fra nu af vil benævne ***O***

Når vi arbejder med grupper på Elliptiske kurver over et endeligt felt F_p er der naturligvis ikke længere tale om en sammenhængende visuel glat kurve , men nærmere en "punktsky"
F.eks kan y^2 mod 23 = x^3 + 4x + 3 mod 23 visualiseres således:



Skalarmultiplikation af punkter er grundlæggende for elliptiske kurvers kryptografiske anvendelse.

Et punkt P multipluceret med et tal k resulterer I et nyt punkt på kurven Q = kP.
Skalar multiplikation udføres ved en kombination af punkt addition og punktfordobling (dublikation)
Eksempel: Q= 11P = 2((2(2P)) + P) + P

Pollards p-1 metode:

Kerne ideen i Lenstras metode tager sit udgangspunkt i Pollards p-1 metode men på en anden gruppe.
Det er derfor relevant først kort at se på Pollards p-1 metode
Fra Fermat's lille sætning ved vi at hvis a ikke er divisibel med p, dvs gcd(a,p)=1 så har vi at

$$a^{p-1} \equiv 1 \bmod p \qquad (1.4)^{1}$$

Ser vi på en gruppe derfineret på $\mathbb{Z}/\mathbb{Z}_p$ har vi at gruppens orden $\varphi(p)=p-1$.

Hvis vi har et tal t så p-1|m har vi ligeledes at

$$a^{m} \equiv 1 \bmod p \qquad (1.5)$$

Idet m=k(p-1) og vi får da at (1.5) er det samme som $a^{k(p-1)} = (a^{(p-1)})^{k} = 1^{k} = 1 \bmod p$
Hvis vi ønsker af faktorisere et sammensat tal N og primtallet p en faktor i tallet og vi har at p ikke deler a, kan vi konkludere[2] at gcd(N,a^m-1) > 1 da vi har at p| a^m-1.
Vi kan altså med passende valg og variationer af a og m håbe at vi rammer et m som er divisibel med p-1, hvor p er primfaktor i N.
I praksis kan vi naturligvis ikke beregne modulus p da det er den ukendte faktor , men vi kan regne modulus N. Ethvert $d = a^m - 1 \bmod N$ som ligger i intervallet $1 < d < N$ er da en faktor i N
Er d=1 varieres t og er d=N varieres a.

[1] Ref[14] s. 26
[2] Ref[14] s. 33

Algoritmen virker især hvis p-1 er flad, dvs består af mange små primtal, og m kan vælges som et produktet af stigende små primtal . Ulempen ved dette er naturligvis at a^m vokser meget hurtigt når m vokser.

Bound

Typisk vælger man en øvre grænse – et såkalt Bound B så vi gennemløber kun de primtal der er mindre end B som deler t.
Hvis vi da har at ordnen af $\mathbb{Z}/\mathbb{Z}_p$; p-1 er B-glat hvis alle faktorer i tallet p-1 er mindre end B.

Og da vi altid har at elementernes orden går op i gruppens orden, p-1, har vi også at hvis gruppens orden er flad så er elementerne i gruppens orden også flad
Så er det sansynligt at B! er et multiplum af p-1. Det betyder at vi kan forvente med en vis sansynlighed at $a^{B!} \equiv 1 \bmod p$

B! er dog meget hurtigt et meget stort tal. Så $a^{B!}$ vil meget hurtigt lave overflow i computeren.
Vi vil derfor typisk beregne dette rekursivt moduleret; $a^{B!} \equiv (a^{(b-1)!})^b \bmod N$ for b=2,3,4,...B.
Modulær exponentiering kan også gøres binært med på hinanden følgende kvadreringer, såkaldt hurtig potensopløftmning[3].
Vi vil derefter beregne $\gcd(a^{B!}-1, N)$ og forhåbentligt ramme et tal større end en, en faktor.
Det er samme princip der anvendes i første fase af den elliptiske kurvemetode.

Eksempel:

Hvis vi for eksempel ønsker at faktorisere tallet N=42037 gør vi altså følgende:
Lad a=3 og m=2*3*5*7*11*13*17=510510. Vi beregner $a^m = 3^{510510} \bmod 42037 = 28467$
Vi ser derfor på gcd(42037,28466) og får 331. Vi regner efter og ser at 42037 =331*127
Havde vi i stedet valgt m=3*5*7*11 får vi 3^(2*3*5*7*11) mod 42037=22840 og gcd(22839,42037)=331. Valgte vi deriomd et mindre m=2*3*5*7 får vi 3^(2*3*5*7) mod 42037=23006. Men gcd(23005,42037)=1 og vi finder derfor ingen divisor i N med dette m.

Det er omkostningsfuldt at arbejde på meget store N og m. Vi kan desuden meget nemt risikere at p-1 ikke er flad og så er metoden ikke effektiv. Ulempen ved metoden er at den kan fejle og vi kan da ikke blot vælge en anden gruppe at arbejde på.. Det er netop hvad Lenstra viste var muligt på grupper defineret over elliptiske kurver.

Elliptisk curve metode:

Ideen er derfor at bruge tilfældige pseudoelliptiske kurver. Fordelen er at hvis en kurve fejler vælger vi blot en ny.
Ved at ændre koefficienterne på den elliptiske kurve ændrer vi også på antallet af punkter på kurven og dermed også på ordnen.

Vi ved fra Hasses sætning at antallet af punkter på kurven $\#E(F_p)$ ligger inde for intervallet:

[3] Ref [14] afsnit 2.3.14 og 2.3.15

$$p + 1 - 2\sqrt{p} < \#E(F_p) < p + 1 + 2\sqrt{p} \quad (1.6)$$

Hvor Pollards p-1 metode er defineret over gruppen $\mathbb{Z}/\mathbb{Z}_p$, er Lenstra's Elliptic Curve udskiftet med gruppen på en elliptisk kurve $E(F_p)$ over et endeligt felt F_p,hvor p er den primfaktor i n vi ønsker at finde. Det forholder sig heldigvis sådan at det ikke er nødvendigt at kende p på forhånd(!) Vi regner i stedet på $E(Z_n)$ da vi har at $E(Z_n) = E(F_p) \oplus E(F_q)$ jf ref [2][4]

Overordnet set er ideen så at der ved summation af punkter P+Q på elliptiske kurver vil være situationer hvor P+Q ikke er defineret. Vi vil være i den situation når vi skal beregne hældningen hvor vi rammer elementer forskellig fra 0 i $\mathbb{Z}_N$ der ikke er invertible. Eksistensen af ikke invertible elementer opstår da vi jo regner på et N der er sammensat og vi regner derfor ikke i en rigtig gruppe, og dermed heller ingen garanti for inverser. Deraf navnet pseudo-elliptiske kurver. Men det er netop denne "fejl"-situation der er interessant , for når vi har fundet et ikke invertibelt element i N har vi i stedet fundet en primfaktor.
Det vil altså sige at vi ønsker at finde et k så

$$kP = \boldsymbol{O} \text{ i } E(F_p) \text{ eller } E(F_q) \quad (1.7)$$

Algoritmen handler da om at dels vælge kurver, dels findet velegnede punkter og dels at producere nye punkter i jagten på at fange et punkt hvor det går "galt" (godt).

Et lille eksempel på dette herunder:

Vi vil gerne finde i faktor p i N=172841 og regner derfor i Z_{172841}
Vi tager en tilfældig elliptisk kurve:
$y^2 = x^3 + 18x + 64$ og et punkt P på kurven [0,8]
Vi sikrer os at diskriminanten er forskellig fra 0;
$d == -(4A^3 + 27B^2) = -4(18^3 + 27 * 64^2) \bmod 172841 = 79986 \neq 0$
Vi finder nye punkter på kurven ved at beregne kP. Dette findes ved at multiplicere og addere med udgangspunkt i kP. Vi skal altså derfor forsøge at finde et k så kP ikke er defineret således at inversion er muligt.
Hvis vi serfor f.eks forsøger os med 3P (=2P+P), skal vi altså beregne først Q=2P= P+P og dernæst Q+P=3P
Vi anvender derfor (1.3) og beregner derfor Q=P+P=[62116,16532] og beregner dernæst Q+P= [148635, 80180]. 3P gav altså ikke bid. Beregner vi derimod 16P som kan beregnes ved fordobling 2P->4P->8P->16P (= 2(2(2(2P)))) får vi 4P= [149112, 87129], 8P= [102968, 91769], men fordoblingen af 8 P går galt og vi får i sted faktoren 563 ud således:
Vi beregner først tangenten m for 2(8P) jf (1.3)
$m=3*(102968)^2 + 18 / 2 * 91769 \bmod 172841 = 162065 / 10697 \bmod 172841$, men dette er ikke defineret da 10697^{-1} ikke findes mod 172841. i stedet har vi at $\gcd(172841,10697) = 563$. – og vi har altså dermed fundet en faktor i N
Vi har dermed altså at punktet 16P er uendelig mod 305 men endelig mod 563 og derfor får vi faktoren 536 ud når vi forsøger at beregne 16P mod (305*563)

[4] Ref [2], Elliptic Curves, Washington side 180

Generelt er det altså sådan at ved hældningsberegning har vi at hældningen x = a/b mod n, det er det samme som kongruensligningen:

$$ax \neq b \bmod N$$

Men det betyder jo at dette har en løsning hvis og kun hvis gcd(a,N)= d går op i b, altså d|b.
Men er gcd(a,N) forskellig fra 1 kan vi ikke finde a^{-1} mod N , da der kun findes en multiplikativ invers hvis gcd(a,N) = 1 og vi kan derfor ikke evaluere hældningen.
Men vi har i stedet fundet en faktor i N, nemlig a og har at N=a*m, hvor m let findes m=N/a.

Hvis ordnen havde været det samme ville vi have fået at gcd ville have været N og vi have ikke fundet en faktor. Men sandynligheden for at ordnen af punktet er det samme mod a og m er meget lille.

Det største problem i praksis vil være at finde et tal k så kP= uendelig modulus en faktor i N
Men med kurver nok er det sansynligt at i hvert fald en af dem giver mulighed for at finde et sådant k, da enhver ny kurve vil have et andet antal punkter på kurven og dermed en anden orden, der giver nye chancer; dette som nævnt i modsætning til Pollards p-1 metode hvor gruppen ikke blot kan udskiftes.
Dette betyder en stor fordel ved anvendelse af elliptiske kurver frem for p-1 metoden ved store N, også selv om regneoperationerne er dyrere på elliptiske kurver

Projektive koordinater Mongomery form

Det er dog ikke effektivt at regne på den korte Weierstrassform da det er meget omkostningsfuldt at regne inversion modulus N og Montgomery formen giver os mulighed for at undgå dette.
Ifølge artikelen erstatter vi istedet (1.1) med den homogeniserede Weierstrass ligning:

$$y^2z = x^3 + axz^2 + bz^3 \tag{1.8}$$

Det betyder at vi i stedet for affine koordinater arbejder med projektive koordinater.
Fordelen ved dette er at vi ikke behøver arbejde med inverser i aritmetiske kurveoperationer, hvor det er tilfældet i når vi anvender den affine form.
Vores beregninger er derfor ret afhængige af valg af coordinatsystem.
Den lange Weierstrass form ser ud som følger

$$y^2 - a_1xy + a_3y = x^3 - a_2x^2 + a_4x + a_6 \tag{1.9}$$

og ikke alle kurver kan skrives på kortere form ved rationelle transformationer, men i artiklen foreslåer de anvendelse af de såkalte Montgomery koordinater der er baseret på projektive koordinater og anfører at variationer af a herunder alligevel giver et tilstrækkeligt stort udbud af klasser af pseudo-tilfældige kurver.. Montgomeryformen ser ud som følger

$$by^2 = x^3 + ax^2 + x \tag{1.10}$$

Eller ved at projicere kurverne over i $P^2(K)$ hvor vi opererer med punktet [X,Y,Z] hvor $Z \neq 0$ korresponderer til det affine punkt (X/Z,Y/Z). Punktet [0,1,0] er punktet ***O*** i uendelig.Det er dog den særlige Montgomery version af projektive koordinater vi anvender hvor vi dropper Y koordinaten, som ifølge Brent ikke er nødvendig. Vi har da punkter af formen [X:Z], hvor x=X/Z
Vi får da ved modifikation af (1.10)

$$by^2z = x^3 + ax^2z + xz^2 \tag{1.11}$$

Vi har nu en ændret diskriminantbetingelse, nemlig: $b(a^2-4)\neq 0$
Vi får da skalarmultiplikation af et punkt $P_1=(x_1:y_1:z_1)$ at $P_n=nP_1$ og $P_n=(x_n:y_n:z_n)$ kan skrives på formen $P_n=(x_n::z_n)$
Vi har følgende additionsformel for $m\neq n$

$$\frac{x_{m+n}}{z_{m+n}}=\frac{z_{|m-n|}(x_m x_n-z_m z_n)^2}{x_{|m-n|}(x_m x_n-z_m z_n)^2} \tag{1.12}$$

Og følgende dublikationsformel

$$\frac{x_{2n}}{z_{2n}}=\frac{({x_n}^2-{z_n}^2)^2}{4x_n z_n({x_n}^2+ax_n z_n+{z_n}^2)} \tag{1.13}$$

Umiddelbart kender vi jo ligeså lidt x_{m+n} som vi kender x_{m-n}. Men ifølge Montgomery [5] kan vi anvende binær metode. I denne situation kan vi så netop starte med m-n=1 og da vil koordinaterne $(x_{m-n}::z_{m-n})$ være $(x_1::z_1)$. Koordinaterne for $(x_k::z_k)$ findes da via dublikation og addition som også vist før.
Eksempel:
$[X_3:Z_3]$ findes ved først at dublikere $[X_1:Z_1]$ til $[X_2:Z_2]$ og addere $[X_1:Z_1]$
$[X_6:Z_6]$ findes ved at dublikere $[X_3:Z_3]$

Tilbage står nu at se på den generelle algoritme, algoritmens effektivitet, generelle startbetingelser, valg af kurve og valg af startpunkt, samt overvejelser om effektivisering af additionsalgoritmer og multiplikationsalgoritmer.

Man opererer i EFC med to faser. Lykkes fase 1 ikke fortsættes med en fase 2.
Den første fase er i grove træk allerede skitseret og fase 2 findes i en række varianter.

Fase 1.

I praksis sætter vi først og fremmest sætter vi en øvre grænse for hvor stort et tal r, $r*P_1=P_r$ r vi ønsker at regne på.
r vælges da ofte som produktet af alle primtalspotenser mindre end en grænse B_1. jf Bound overvejelser under Pollards p-1 metode.
Vi har da netop at hvis vi rammer en ikke defineret modulær invers, der hvor division ikke er mulig i hældningsberegningen at det er tegn på at der findes et p|N så
kP=***O*** hvor

$$kP=\prod_{p_i^{a_i}\leq B_1} p_i^{a_i} \tag{1.14}$$

[5] Ref[4] side 261

Vi behøver ikke beregne r direkte da der gælder at $\ln(r) \rightarrow B_1$ når $B_1 \rightarrow \infty$.

Det betyder at vi kan beregne $(x_r :: z_r)$ i O(ln (r))=O(B_1) operationer. Dette kan ifølge artiklen reduceres yderligere til 11/ln2* B_1 operationer, eller hvis z_1 =1 så 10/ln2* B_1 operationer
Fase 1 afsluttes med et tjek om P_r = ***O,*** ved at beregne gcd(z_r,N)
Er denne forskellig fra 1 eller N har vi fundet en faktor. Er dette ikke tilfældet går vi videre med fase 2 før eventuelle nye kurver bringes på banen.
I praksis vælges grænsen B_1 f.eks som 10^6 hvis vi skal finde faktorer op til 35 decimalcifre (godt 100 bit).

Kurvekonstruktion og startpunkt

Det viser sig til[6] at ved anvendelse af Montgomerys form (1.10) vil gruppens orden altid være divisibel med 4, men dette kan forbedres yderligere, således at vi ved at gruppens orden er divisibel med 12.
Dette gøres ved at udvælge a i ligningen (1.10) så den opfylder en række betingelser:

$$a = C(\sigma) = \frac{(v-u)^3(3u+v)}{4u^3v} - 2 \qquad (1.15)$$

Hvor der samtidig gælder at u er på formen $u = \sigma^2 - 5$ og v på formen $v = 4\sigma$
Vi kan da vælge nye kurver E_σ ved blot at vælge et tilfældigt σ
Startpunktet vælges så $x_1 / z_1 = u^3 / v^3$

Fase 2

I Fase 2 opererer vi med en højere grænse B_2 . Moderne implementationer af ECM viser at man højest bør bruge mellem ¼ og ½ af beregningstiden på fase 2 og dette giver i praksis at B_2 bør være af størrelsesordnen 100 B_1 se ref[7]

Standard fortsættelsen

Vi anvender koordinater på formen (1.10) og arbejder nu ikke med en, men med to grænser B_1 og B_2 idet vi nu antager at der kan findes en faktor q større end B_1 men mindre end B_2 , da vi altså må formode at ordnen af gruppen over kurven E(F_p) ikke var B1-glat, og at vi derfor har et q så ordnen kan skrives som

$$\#E(F_p) = q * \prod_{p_i^{a_i} \le B_1} p_i^{a_i} \qquad (1.16)$$

B_2 vælges så den er meget større end B_1, feks som nævnt 100* B_1. Det betyder at forskellen mellem dem går mod B_2 .

[6] I følge Suyama, jf ref[1]s 434 og ref[3] s 341
[7] Prime Numbers, Crandall og Pomerance s343, ref[3]

Vi forudsætter at vi ikke fandt en primfaktor i fase 1 og vi har derfor beregnet et punkt Q så således at $Q = kP \neq \boldsymbol{O}$

I denne fase beregner vi så punkterne sQ for alle s mellem de to grænseværdier B_1 og B_2 . Dvs: $sQ = (x_{rs} :: z_{rs})$ for alle s, $B_1 < s < B_2$. Vi har som i fase 1, at vi har fundet en faktor hvis gcd(z_{rs} ,N) er forskellig fra 1 eller N.

Der er dog ikke nogen beregningsmæssig gevinst da denne metode tager O(log s) =O(B2) gruppe operationer.

Men dette kan optimeres ved at anvende en tabel med forudberegnede værdier i et afgrænset område der kan anvendes som mellemregninger til videre beregninger.

Brent anfører at hvis vi vælger en øvre grænse for tabelværdien D og beregner alle punkterne 2dQ for d gennemløber primtallene mellem 0 og D Man gemmer altså værdien for nogle punkter permanent $R_i = d_i Q$, og man kan derefter hurtigt beregne øvrige punkter ved passende kombinationer af punkterne i tabellen.

I tabellen beregner vi $\min(s_1 + 2D, s_2)Q$ for et ulige s_1 og s_2 er det næstefølgende primtal. Dette kan gøres med kun en operation. Det betyder at vi kan beregne alle sQ for en række af værdier mellem B_1 og B_2 med kun en operation pr punkt. Crandall og Pomerance[8] angiver algoritmen for tabelværdierne således, hvor q_0 er det mindste primtal større end B_1:

$q_o Q, (q_o + d_0)Q, (q_o + d_0 + d_1)Q, (q_o + d_0 + d_1 + d_2)Q...$ hvor d_i er forskellen på hinanden følgende primtal efter q_0

Tabellen vil betyde datalagring i størrelsesordnen O(log N log B_2) bit.

Man definerer et B_3 som $\pi(B_2)$-$\pi(B_1)$, dvs antallet af primtal mellem B_2 og B_1. Dette angiver Brent er cirka lig med B_2 /log B_2 .Tidsforbruget i Fase 2 er da proportionelt med B_3 og er reduceret fra O(B2) til O(med B_2 /log B_2).

Forbedret standard fortsættelse

Den forbedrede version har især øje for optimering af algoritmen i standardversionen.

Den overordnede ide i den forbedrede standard version er at det er muligt at beregne gcd(z_{rs}, N) uden at beregne sQ,

Metoden bygger også på forudberegnede værdier, og på anvendelse af Montgomery koordinater i igennem alle beregninger.

Vi forudberegner 2dQ for d liggende i intervallet 0<d<D som før.

Herefter beregner vi mQ, hvor m=1,2D+1,4D+1.... Hvis vi da har et primtal s=n+m, og et lige n liggende i intervallet 0<n<2D kender vi altså både $mQ = (x_{mr} :: z_{mr})$ og $\pm nQ = (x_{nr} :: z_{nr})$.

Og hvis vi har at sQ=$\boldsymbol{O}$ så betyder det at $mQ = \pm nQ$. Vi behøver derfor blot teste for største fælles divisor med krydsproduktet: $x_{mr} z_{nr} - x_{nr} z_{mr}$, altså om

$\gcd(x_{mr} z_{nr} - x_{nr} z_{mr}, N) \neq 1$

[8] Ref[3] s 340

Da vi beregner mQ on the fly behøver vi ikke gemme værdierne for denne. Der er ligeledes en række optimeringer på algoritmen, som især handler om at reducere antallet af multiplikationer modulus N pr s.
Det kan vi ved at beregne produktet af koordinatdifferencerne over mange m og n.
Det kan reducere antal multiplikationerne til 3 pr s. Vi kan altså med fordel gemme det akkumulerede produktet af koordinatdifferencerne over mange m og n $\prod_{n,m}(x_{mr}z_{nr} - x_{nr}z_{mr})$, i forventning om en største fælles divisor mellem et sådant produkt og N.
En yderligere optimering [9] kan foretages ved at bemærke at krydsproduktet kan beregnes således:

$$x_{mr}z_{nr} - x_{nr}z_{mr} = (x_{mr} - x_{nr})(z_{mr} - z_{nr}) + x_{nr}z_{nr} - x_{mr}z_{mr} \quad (1.17)$$

Vi kan da gemme forudberegnede værdier af $x_{nr}, z_{nr}, x_{nr}z_{nr}$ og anvende $x_{mr}, z_{mr}, x_{mr}z_{mr}$ efterhånden.

Det vil være en afvejning af tilgængelig plads at gemme på i forhold til beregningstid undervejs. Men generelt er pladsbehovet ikke meget større end for standard fortsættelsen.
Det er muligt at få antallet ned på 1 ved forlods at normalisere z_{nr} til at være 1. Det kræver dog yderligere forudberegning.

Fødselsdags paradox fortsættelsen

Vi bruger begrebet fødselsdagsparadoxet til at beskrive den periode der går før vi kan forvente at en værdi i en mængde gentager sig med en sansynlighed på over 50% . Denne periode går asymptotisk mod $\sqrt{\frac{\pi N}{2}}$ hvor N er udfaldrummet. I den klassiske udgave som har givet navn til begrebet er N= 365 dage og resultatet 23.9 er svaret på hvor mange personer der skal samles før sansynligheden er over 0,5 for at to har fødselsdag samme dag.

Fødselsdagsfortsættelsen går ud på at man konstruerer koordinater for to sæt pseudotilfældige punkter og sammenligner disse.

Vi tager altså og genererer en antal pseudo-tilfældige multiplum af Q, hvor Q er det punkt vi har fra fase 1. Vi genererer i dette tilfælde Q_j således:

$$Q_j = q_j{}^e Q \quad (1.18)$$

For j=1,...T, hvor T også er størrelsen af tabellen. e sættes til at være produktet af små primtal.
Der er en række forskellige måder at genererer disse pseudo-tilfældige punkter.
F.eks foreslår Brent i en tidligere artikel[10] at der i stedet vælges en rekursiv funktion der sender Q_{j+1} over i følgende to forskellige værdier, hver med sansynligheden en ½
; $Q_{j+1} = Q_j{}^2, Q_{j+1} = Q_j^2 * Q$.
Men anvendes (1.18) genereres herefter efterfølgende en ny mængde pseudotilfældige punkter med

$$\overline{Q}_k = \overline{q}_k{}^e Q \quad (1.19)$$

Hvor $\overline{q}_k \neq q_j$. $\overline{q}_k$ kan f.eks vælges som $\overline{q}_k = 2^k$
Vi tjekker da for hvert punkt om $\overline{Q}_k = Q_j$ og er det tilfældet, kan vi finde en faktor i N.

[9] Ref [3] s 342
[10] Some Integer Factorization Algorithms using Elliptic Curves, Brent ref[5]

Vi bruger i denne version at sansynligheden for at dette indtræffer er stor selv med relativ få punkter.

Men ellers altså i princippet det samme som forbedret standard version, men den forskel at vi her forsøger at optimere sansynligheden ved hjælp af ”parring” af pseudotilfældige punkter hvorimod vi systematiserer dette i forbedret standard version
Der er ifølge ref[1] ikke den store forskel i hverken performance eller pladskrav på de to versioner.

Pseudokode og implementationsovervejelser

Den første del af ECM med anvendelse af den korte Weierstrassform kan i grove træk implementeres efter følgende pseudokode
(1.20)

```
//1. vælg grænseværdi B
B=1000000;
//skal vi faktorisere det angivne 56 decimal cifrede tal kan vi vælge B1 //=10^6 for at
finde faktorer op til 35 decimaler

//2. Ny kurve E(a,b,N) og punkt P=(x,y)
Random x,y,a (0..N-1);
b=(y^2-x^3-ax) mod N;
g=gcd(4a^3+27b^2,N);
if (g=N) → Ny Kurve
if(g>1) – returner g //EN FAKTOR ER FUNDET
E(a,b,N) , P=(x,y)

//3. Multiplikation af primtalspotenser
for (1<i<π(B) ){
   Find det største a_i så a_i^k_i < B
   for(1<j<a_i ){
   Q = p_i P , //her anvendes effektive dublikations og additionsformler
          stop if d^-1 mod N ikke er defineret
                 g=gcd(d,N)
                 returner g
    }
 }
//4. returner ” ingen faktor er fundet”
B++ (vælg større B);
Ny kurve;
Eller
Goto fase 2(Q)
```

Vi har til dette brug for effektive implementationer af BigInteger, gcd(), modulærinvers: $d^{-1} \bmod N$, effektiv binær potensopløftning, samt en tabel over $\pi(B)$, antallet af primtal op til en grænseværdi B.
Dublikationsfunktioner og additionsformler er oven over givet ved (1.2) og (1.3).
Arbejder vi derimod med projektive Montgomery koordinater, uden Y er de givet ved (1.12) og (1.13).

Dette kan udspecificeres lidt nærmere sådan her ifølge Crandal og Pomerance:

Plusfunktion($[X_m:Z_m],[X_n:Z_n],[X_-:Z_-]$){

$X_+ = Z_-((X_m X_n - AZ_m Z_n)^2 - 4B(X_m Z_n + X_n Z_m + CZ_m Z_n)Z_m Z_n)$

$Z_+ = X_-(X_m Z_n - X_n Z_m)^2$

Returner $[X_+,Z_+]$

}

Dublikationsfunktion(){

$X_+ = (X_m{}^2 - AZ_m{}^2)^2 - 4B(2X_m + CZ_m)Z_m{}^3$

$Z_+ = 4Z_m(X_m{}^3 + CX_m{}^2 Z_m + AX_m Z_m{}^2 + BZ_m{}^3)$

Returner $[X_+,Z_+]$

}

Hvilket er en smule mere generel end de i teksten angivne, da der her er anvendt en elliptisk kurve af formen : $gy^2 = x^3 + Cx^2 + Ax + B$. Hvis vi sætter B=0, A=1, C=a og g=b får vi en præcis en ligning af formen (1.10). og med B=0 falder flere led væk og funktionerne er identisk med dublikations og additionsformlerne i (1.12) og (1.13).

Igen inspireret af ref[3] er en lidt grov pseudokode for den forbedrede standard verson er forsøgt skitseret herunder.

(1.21)

//Vælg grænseværdier for B1 og B2, samt en værdi for D
B1=100000;
B2=100B1;
D=100

//sikre at 2 og 3 ikke er divisorer i tallet
Tjek Gcd(6,N)=1

//Vælg tilfældig kurve E_σ
Random σ (6..N-1));
$u = (\sigma^2 - 5) \bmod N$;
$v = 4\sigma \bmod N$
$C = (((v-u)^3(3u+v)/(4u^3 v) - 2) \bmod N$

//vælg startpunkt Q [X:Z]
$Q = [u^3 \bmod N : v^3 \bmod N]$

//Fase 1
for (1<i< $\pi(B)$){
Find det største a så $p_i{}^a < B_1$
$Q = p_i{}^a Q$, // $[X_Q:Z_Q]$
}

g=gcd(Z_Q ,N)

if (1<g<N) returner g //FAKTOR FUNDET!

Ellers → fase 2

//Fase 2

$S_1 = dublikationsfunktion(Q)$;

$S_2 = dublikationsfunktion(S_1)$

//beregn alle 2dQ

For(1< d< D){

If (d>2)

$S_d = plusfunktion(S_{d-1}, S_1, S_{d-1})$

//gem også koordinat produkterne i Sd

$\beta_s = X_{S_d} * Z_{S_d} \bmod N$

}

g=1;

B=B1-1

//midlertidige punkter

T=[B-2D]Q;

R=[B]Q;

//loop i intervallet mellem B1 og B1 med step 2D

For(r=B,r<B2,r=r+2D){

//beregn koordinat produktetrne i S_R

α=X_R*Z_R mod N

//loop over primtallene i intervallet i intervallet mellem r og r+2D

For(q=r+2;q<r+2D;q++(kun primtal)){

δ=(q-r)/2

g=g((X_R -X_{S_δ})(Z_R +Z_{S_δ})-α+β_δ)mod N

}

(R,T)=(dublikationsfunktion(R,S_D,T),R)

}

g=gcd(g,N)

if (1<g<N) returner g //FAKTOR FUNDET!

Ellers → //Vælg tilfældig kurve E_σ

Vi gemmer altså XZ produkterne her for hvert multiplum af 2dQ. VI nærmer os aymptotisk til en faktor g her med to multiplikationer mod N pr kandidat.

Ifølge Silverman og Wagstaff[11] der testede perfromance af Lenstra ECM på baggrund af Montgomerys optimeringer vil vi i tilfældet med det 56 cifrede tal t, hvis vi forsøger at finde en faktor med onkring 25 cifre

[11] A practical Analysis of Elliptic Curve Factoring Algorithm, Silvermann og Wagstaff ref [6]

skulle sætte B_1 optimalt sættes til 66596 og B_2 til $2{,}6*10^6$, vi vil i gennemsnit skulle igennem 376 kurver og forvente et tidsforbrug på $3{,}5*10^7$

Forudsætningerne for dette tidsforbrug er følgende beregning: $T=L(B_1+(B_2-B_1)/K)$ hvor L er antallet at kurver og K er antallet af gruppeoperationer pr kurve.I ovenstående er K sat til 100. Alle disse beregninger er dog mere end 10 år gamle. Men tabellen det giver dog en fornemmelse af forholdstal idet vi kan se at hvis vi starter lidt mere forsigtig med at søge efter faktorer i området 19 cifre, vil vi kunne gå helt ned på et B_1=9004 og B_2 = 405180 og forvente at skulle bruge 106 kurver.

Da langt de fleste operationer til ECM anvendes på multiplikationer modulus N er der meget at hente i af effektivisere der

Brent angiver i artiklen ref[1] en metode til beregning af antallet af multiplikationer mod N:

$$W(p) = \exp(\sqrt{(2+o(1))\log p \log\log p}) \tag{1.22}$$

Hvor p er den forventede faktor. Den forventede runtime er da :

$$T(p,N) = M(N)*W(p) \tag{1.23}$$

Hvor M(N) er den tid der skal anvendes pr multiplikation modulus N.
Brent angiver at ved forventet faktor på 20 cifre skal optimalt forventes at $\log_{10}W=7{,}35$. Dvs W er cirka 22,5 millioner multiplikationer, eller $2{,}25*10^7$ Er vi oppe i en faktor med 40 cifre skal vi i stedet forvente $\log_{10}W= 11.49$ log_10W= 11.49, dvs W er omkring 310 milliarder, eller $3{,}10*10^{11}$.
Det ses af tidberegningsalgoritmen er det altså ligeså vigtigt at reducere altallet af multiplikationer pr operation, f.eks pr punktaddition, som det er at anvende stor computerkraft.

Andre implementationer

Uden at kende alle de præcise algoritmer for implementarionerne i forskellige matematikprogrammer er det dog stadig interessant at sammenligne hvor nemt eller svært de har ved at faktorisere tallet
t=490398573077084388736551514361406371199542212048527032 59.
Testen er foretaget på en Pentium4, 2 GHz med 1GbRam.
En hurtig test viste at den indbyggede elliptiske factoriseringsalgoritme i MuPad gav det højst overraskede svar ’1’ på spørgsmålet:
numlib::ecm(490398573077084388736551514361406371199542212048527032 59);
Dens mere uspecificerede faktoriseringsfunktion ifactor(t) gav dog et mere korrekt svar på et par minutter:
Mathematica 4.0 svarer korrekt med FactorIntegerECM[t] på 26 sekunder
Maple 10.0 derimod bruger faktisk 3897sekunder, eller godt en TIME på at udføre følgende kommando: ifactor(t,lenstra); Der må derfor være tal om en en decideret fejl.
Dette blev bekræftet da jeg efterfølgende installerede Maple 9 på samme maskine og denne klarede t på 136 sekunder.
Et kik på kildekoden vil afsløre en enkelt afvigelse mellem de to implementationer der ligner en tastefejl. Kildekoden kan ses ved at kalde:

```
>interface(verboseproc = 3):
>eval(ifactor);
```

Et eksempel på implementation af en ECM funktion i Maple, af Roman Pearce, tilgængelig på MapleSofts website[12] (samme som anvendes i Jørgen Brandts noter) giver op over for opgaven. Det er et demonstrationseksempel der ikke er optimeret på nogen at de i dette papir diskuterede måder, den indeholder kun grundfunktionaliteten i ECM, svarende til pseudokoden i (1.20). Det er da heller ikke endnu lykkedes at gennemføre en ECM på det, i dette tilfælde, relevante tal t, uden at programmet er gået ned.

Derimod ligger der en JAVA implementation af Dario Alejandro Alpern fra 2005 tilgængelig på http://www.alpertron.com.ar/ECM.HTM, inklusiv kildekode der knækker t på 8 sekunder. Programmet oplyser samtidigt at det finder svaret ved den 28. elliptiske kurve efter 4089782 modulære multiplicationer og at den har anvendt følgende grænseværdier som 'bounds': B_1=50000; B_2 =5000000.

En moderne implementation der altså er langt mere effektiv end gennemsnitligt antal kurver forventet ifølge ref[6] og altså også en version der overlegent udkonkurrer matematikprogrammerne.
Implementationen er baseret på Montgomery modellen og bruger i fase 2 forbedret standard fortsættelse, men indeholder derud over yderligere en lang række avancerede funktionaliteter og test.
I tabellen sammesteds er anført forventet kurve antal for faktorer mellem 15 og 20 cifre til af være mellem 25 og 90 kurver. Samtidig er dog angivet at optimal værdi for B_1 i dette område er langt lavere end anvendt, nemlig mellem 2000 og 11000.
Tallet t blev faktoriseret til
490398573077084388736551514361406371199542212048527032 59=
28823037615171171 7*17014118346046923173168730371588410 5727, hvilket iøvrigt også er (2^58 - 27)*(2^127 - 1)

Perspektiv

Der er en lang række varianter og optimeringsmuligheder i ECM, også ud over de her nævnte, men samlet set er ser ECM ud til at være bedst til at afsløre middelstore primfaktorer, også selv om N selv er meget stor. Ifølge Washington s. 182 er algoritmen generelt succesfuld hvis vi skal finde faktorer i sammensatte tal N hvor faktoren er under 10^{40} .
I praksis er det sjældent at fange primtal med ECM på omkring 30 decimalcifre og endnu sjældnere i størrelsesordnen 40 decimalcifre.[13]
Er faktorerne små, er andre mindre omkostningsfulde metoder mere effektive.

Hvad betyder det så at den elliptiske kurvemetode er så fremragende til at faktorisere endog meget store tal? Jeg det stiller jo altså dermed større krav til nøglestørrelser og komplexitet i de kendte krypteringsalgoritmer.
Ved kryptografiske systemer vælger man da i dag også typisk sammensatte tal N=pq hvor p og q har mindst 75 decimalcifre hver[14].

[12] http://www.maplesoft.com
[13] s 347 ref [3]
[14] s. 182 ref [2]

Kildeoversigt

[1] Factorization of the tenth Fermat number (1999), Richard P Brent
[2] Elliptic Curves, Number Theory and Cryptography(2003) af Laurence C. Washington
[3] Prime Numbers, A Computational Perspective 2.ed (2005) af Richard Crandall og Carl Pomerance.
[4] Speeding the Pollard and Elliptic Curve Method of Factorization(1987) af Peter L. Montgomery
[5] Some Integer Factorization Algorithms using Elliptic Curves(1985), Richard P Brent
[6] A Practical Analysis of the Elliptic Curve Factoring Algorithm(1993), Silvermann, Wagstaff

Yderligere baggrundsmateriale der er konsulteret

[7] Factoring Integers with Elliptic Curves(1987), H.W.Lenstra
[8] Theorems on factorization and primality testing (1974), J.M.Pollard
[9] Factoring by electronic mail (1990), Lenstra, Manasse
[10] An Improved MonteCarlo Factorization Algorithm(1980), Richard P. Brent
[11] Guide to Elliptic Curve Cryprography (2004), af Hankerson, Menezes og Vanstone
[12] The state of Elliptic Curve Cryptography(2000), Koblitz,Menezes og Vanstone
[13] Integer Factoring(2000), Lenstra
[14] Algebra med Kryptografiske anvendelser(2005) Hans Anton Salomonsen
[15] Concrete abstract algebra(2003), Niels Lauritzen
[16] The art of computer science, Seminumerical Algorithms 2. ed.(1981) af Knuth,