Faktorisierungsalgorithmen

„$(p-1)$-Algorithmus" von Pollard: ist speziell dann effizient, wenn die zu faktorisierende Zahl n einen Primfaktor p besitzt, so dass $p-1$ in ausschließlich „kleine" Primfaktoren zerfällt.

Genauer: $p-1$ ist Teiler von $B!$ für eine nicht zu große Zahl B.

<u>Beispiel:</u> Sei $p = 97$ ein Primfaktor von n.

Dann ist $p-1 = 96 = 2^5 \cdot 3$:

$$p-1 = 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 3$$

$$B! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6 \cdot 7 \cdot 8$$

also $B = 8$

In diesem Fall gilt: $a := 2^{B!} = 2^{(p-1)\cdot k} \equiv 1^k \equiv 1 \pmod p$

Somit: $a-1 \equiv 0 \pmod p$. Dann liefert

$$ggT(\underbrace{a-1}_{= l\cdot p}, \underbrace{n}_{= p\cdot q}) = p$$

einen nicht-trivialen Faktor von n.

Idee: Man probiert immer größere B-Werte aus.

Angenommen, man hat schon $a = 2^{B!} \bmod n$ ausgerechnet, aber $ggT(a-1, n)$ hat noch keinen Faktor von n geliefert, so muss B auf $B+1$ erhöht werden:

$$a_{neu} := 2^{(B+1)!} = (2^B)^{B+1} = (a_{alt})^{B+1}$$

(alles modulo n)

Die Berechnung von a_{neu} aus a_{alt} geschieht also mittels $a := modexp(a, B+1, n)$.

Dies ergibt folgenden Algorithmus:

Eingabe: n
Wähle eine Basiszahl a (z.B $a=2$)
$B := 1$;
<u>loop</u> $B := B+1$;
$a := modexp(a, B, n)$;
$g := ggT(a-1, n)$
<u>if</u> $(1 < g < n)$ <u>then</u> <u>output</u> g

Zahlenbeispiel: $n = 5141 = \underbrace{53}_{q} \cdot \underbrace{97}_{p}$

Es gilt: $p-1 = 96 = 2^5 \cdot 3$ (gut!)

$q-1 = 52 = 2^2 \cdot 13$

Starte mit $a=2$ und $B=1$

B	a	ggT(a-1, n)
1	2	
2	4	1
3	64	1
4	2133	1
5	3685	1
6	4075	97 $\Leftarrow$

Worst-Case Szenario für $(p-1)$-Algorithmus:

Sei $n = p \cdot q$, $p, q \approx \sqrt{n}$

$p-1 = 2 \cdot p'$, $q-1 = 2 \cdot q'$

p, q, p', q' sind Primzahlen. D.h. n ist Produkt zweier (verschiedener) <u>sicherer</u> Primzahlen.

In diesem Fall ist B von der Größenordnung $\sqrt{n}$. Dementsprechend: Laufzeit $O(\sqrt{n})$ wie bei naivem Algorithmus.

Fermat-Faktorisierung

... ist im worst-case auch $O(\sqrt{n})$ laufzeitmäßig. Enthält aber eine gute <u>Grundidee</u>, die allen modernen Faktorisierungsalgorithmen zugrunde liegt.

Sei $n = p \cdot q$ (alles ungerade Zahlen) $p < q$

Setze $x := \frac{p+q}{2}$ und $\frac{q-p}{2} =: y$

$$\Longrightarrow \quad n = \underbrace{(x+y)}_{=q} \cdot \underbrace{(x-y)}_{=p} = x^2 - y^2$$

Differenz zweier Quadrate

Sobald man (durch Probieren, etc.) n als Differenz zweier Quadratzahlen darstellen kann ($n = x^2 - y^2$) hat man Faktorisierung von n gefunden: $n = (x+y) \cdot (x-y)$.

Eingabe: n

$x := \lceil \sqrt{n} \rceil$

$z := x^2 - n$

<u>loop</u> if z ist Quadratzahl, also $z = y^2$

<u>then</u> output $(x-y)$

$x := x+1$

$z := z + 2x - 1$

Begründung für $z := z + 2x - 1$:

Da man x_{alt} auf $x_{neu} = x_{alt} + 1$ erhöht hat, ergibt sich:

$$z_{neu} = (x_{neu})^2 - n = (x_{alt}+1)^2 - n$$
$$= (x_{alt})^2 + 2x_{alt} + 1 - n$$
$$= \big(\underbrace{(x_{alt})^2 - n}_{= z_{alt}}\big) + 2\underbrace{(x_{alt}+1)}_{= x_{neu}} - 1$$

<u>Beispiel</u>: $n = 23 \cdot 53 = 1219$

Initialisierung $\begin{cases} x := \lceil \sqrt{n} \rceil = 35 \\ z := x^2 - n = 6 \quad \text{(kein Quadrat)} \end{cases}$

1. loop $\begin{cases} x := x + 1 = 36 \\ z := z + 2x - 1 = 77 \quad \text{(kein Quadrat)} \end{cases}$

2. loop $\begin{cases} x := x + 1 = 37 \\ z := z + 2x - 1 = 150 \quad \text{(kein Quadrat)} \end{cases}$

3. loop $\begin{cases} x := x + 1 = 38 \\ z := z + 2x - 1 = 225 \quad \text{(ist Quadratzahl: } 225 = 15^2 \text{)} \end{cases}$

$$\Longrightarrow \left.\begin{array}{l} x - y = 38 - 15 = 23 \\ x + y = 38 + 15 = 53 \end{array}\right\} \text{Faktorisierung von } n.$$

Fermat-Faktorisierung ist nur effizient, wenn p und q <u>sehr</u> nahe beinander liegen.

Schon bei $p \approx \frac{\sqrt{n}}{2}$ und $q \approx 2 \cdot \sqrt{n}$

durchläuft x die Werte $\lceil \sqrt{n} \rceil, \lceil \sqrt{n} \rceil + 1, \ldots, \frac{p+q}{2}$
$\approx 1{,}25\sqrt{n}$

Laufzeit also $O(\sqrt{n})$ wie bei naivem Algorithmus.

Quadratisches Sieb - Methode

Die Idee mit den 2 Quadratzahlen x^2, y^2 ist gut! Verallgemeinerung der Fermat-Faktorisierung: Finde x, y so dass

$$x^2 \equiv y^2 \pmod{n}$$

Dann gilt: $x^2 - y^2 = k \cdot n = (x+y) \cdot (x-y)$

Dann liefert $ggT(x-y, n)$ (bzw. $ggT(x+y, n)$) evtl. einen nicht-trivialen Faktor von n (außer wenn $x \equiv \pm y \pmod{n}$).

Wähle hierzu Zufallszahlen $x < n$, berechne $x^2 \bmod n$ und analysiere, ob sich diese Zahl

faktorisieren lässt. Speichere hierzu eine „Faktorbasis" $\{\underset{\substack{\parallel \\ p_1}}{2}, 3, 5, 7, 11, \dots, p_k\} =: \mathcal{B}$
und versuche, ob

$$x^2 \equiv \prod_{i=1}^{k} p_i^{e_i} \pmod n$$

Finde nun „viele" solche Kongruenzen:

$$(x_1)^2 \equiv p_1^{e_{1,1}} \cdot p_2^{e_{1,2}} \cdots p_k^{e_{1,k}} \pmod n$$

$$(x_2)^2 \equiv p_1^{e_{2,1}} \cdot p_2^{e_{2,2}} \cdots p_k^{e_{2,k}} \pmod n$$

$$\vdots$$

$$(x_m)^2 \equiv p_1^{e_{m,1}} \cdot p_2^{e_{m,2}} \cdots p_k^{e_{m,k}} \pmod n$$

Gesucht wird nun eine Teilmenge $I \subseteq \{1, \dots, m\}$
so dass

$$\left(\prod_{i\in I} x_i\right)^2 = \prod_{i \in I} (x_i)^2 \equiv p_1^{\sum_{i\in I} e_{i,1}} \cdot p_2^{\sum_{i\in I} e_{i,2}} \cdots p_k^{\sum_{i\in I} e_{i,k}}$$

Hierbei sollten die Exponenten $\sum_{i\in I} e_{i,l}$ alles
<u>gerade Zahlen</u> sein! $(l = 1, \dots, k)$

Wenn sich solches I finden lässt, dann ist die rechte Seite der Kongruenz eine Quadratzahl. D.h. das gesuchte y ist

$$y = p_1^{(\sum_{i\in I} e_{i,1})/2} \cdot p_2^{(\sum_{i\in I} e_{i,2})/2} \cdots p_k^{(\sum_{i\in I} e_{i,k})/2}$$

<u>Zahlenbeispiel</u>: $n = 19 \cdot 29 = 551$

Die Faktorbasis sei $\{2, 3, 5\}$.

Wir finden z.B. folgende Kongruenzen:

1.) $34^2 \equiv 2 \cdot 3^3 \pmod n$

2.) $52^2 \equiv 2^2 \cdot 5^3 \pmod n$

3.) $55^2 \equiv 2 \cdot 3^3 \cdot 5 \pmod n$

4.) $102^2 \equiv 2 \cdot 3^5 \pmod n$

Die Auswahl $I = \{1, 4\}$ liefert gerade Zahlen in den Exponenten:

$$34^2 \cdot 102^2 \equiv 2^2 \cdot 3^8 \pmod n$$

also $x = 34 \cdot 102$ und $y = 2^1 \cdot 3^4$

Dieses I liefert allerdings keine Faktorisierung:

$$\mathrm{ggT}(34\cdot 102 - 2^1\cdot 3^4,\ 551) = 551$$

Aber es geht mit $I = \{2,3,4\}$:

$$52^2\cdot 55^2\cdot 102^2 \equiv 2^4\cdot 3^8\cdot 5^4 \pmod{n}$$

also $x = 52\cdot 55\cdot 102$ und $y = 2^2\cdot 3^4\cdot 5^2$

Dies ergibt $\mathrm{ggT}(x-y,\ n) = \underline{\underline{29}}$.

Wie kann man ein geeignetes $I \subseteq \{1,\dots,m\}$ finden? (Es gibt 2^m Möglichkeiten!)

$\Rightarrow$ Aufstellen eines linearen Gleichungssystem über $\{0,1\}$ als Zahlenbereich.

$$\text{Setze} \quad \lambda_{i,j} = e_{i,j} \mod 2 \qquad \begin{pmatrix} i=1,\dots,k \\ j=1,\dots,m \end{pmatrix}$$

Bei obigem Zahlenbeispiel haben wir die Exponenten:

$$(e_{i,j}) = \begin{pmatrix} 1 & 3 & 0 \\ 2 & 0 & 3 \\ 1 & 3 & 1 \\ 1 & 5 & 0 \end{pmatrix}$$

Dies ergibt $(\lambda_{i,j}) = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix}$

Die (Zeilen-) Auswahl I sollte so sein, dass sich in jeder Spalte die (modulo 2)-Summe 0 ergibt. Daher kann man ein Gleichungssystem (über $\{0,1\}$) aufstellen:

$$\begin{aligned} a_1 + a_3 + a_4 &= 0 \\ a_1 + a_3 + a_4 &= 0 \\ a_2 + a_3 &= 0 \end{aligned}$$

Jede Lösung $(a_1, a_2, a_3, a_4) \in \{0,1\}^4$ liefert einen Kandidaten für I.

Kann effizient mit Gauß-Vefahren berechnet werden.

Algorithmen für den Diskreten Logarithmus

<u>gegeben</u>: Primzahl n; Primitivwurzel a modulo n
sowie $y \in \mathbb{Z}_n^*$

<u>gesucht</u>: x so dass $y \equiv a^x \pmod{n}$

<u>Naiver Algorithmus</u>: <u>for</u> $x := 1$ <u>to</u> $n-1$ <u>do</u>
<u>if</u> $y = a^x \bmod n$ <u>then</u>
output x

Laufzeit $O(n) = O(2^m)$; $m =$ Anzahl Bits

Babystep-Giantstep Algorithmus:

$$y = a^x = a^{x_1 \cdot \lceil \sqrt{n} \rceil + x_2} \quad \text{wobei } x_1, x_2 \leq \lfloor \sqrt{n} \rfloor$$

$$= \underbrace{\left(a^{\lceil \sqrt{n} \rceil}\right)^{x_1}}_{\text{Giantsteps}} \cdot \underbrace{a^{x_2}}_{\text{Babysteps}}$$

Umgeformt:

$$y \cdot (\underbrace{a^{-1}}_{=:u})^{x_2} = (\underbrace{a^{\lceil \sqrt{n} \rceil}}_{=:v})^{x_1}$$

Methode: Berechne zunächst u und v.

Erzeuge und speichere die Liste

$$\mathcal{L}_1 = \{ (x_1, v^{x_1}) \mid \underbrace{x_1 = 0, 1, \dots, \lfloor \sqrt{n} \rfloor}_{\text{for-Schleife}} \}$$

in einer effizienten Datenstruktur (Suchbaum oder Hashtabelle) so dass es effizient möglich ist, ~~nach~~ ~~einem~~ bei Eingabe von z festzustellen, ob ein Paar (x_1, z) in der Liste vorhanden ist; und wenn ja, das x_1 auszugeben.

Als Nächstes durchlaufe (bzw. erzeuge) die Elemente der Liste

$$\mathcal{L}_2 = \{ (x_2, y \cdot u^{x_2}) \mid x_2 = 0, 1, \dots, \lfloor \sqrt{n} \rfloor \}$$

und teste jedesmal, ob $y \cdot u^{x_2}$ in $\mathcal{L}_1$ vorhanden ist. Sobald die Antwort ja ist und ein x_1 zurückgeliefert wird, ist die Lösung (x_1, x_2) bzw. $x = x_1 \cdot \lceil \sqrt{n} \rceil + x_2$ gefunden.

<u>Laufzeit</u>: $O(\sqrt{n}) = O(2^{m/2})$. Ebenso Speicherplatz: $O(\sqrt{n})$

Zahlenbeispiel: $n = 227$ (Primzahl)

$a = 5$ (Primitivwurzel mod 227)

$y = 86$ ($= 5^{137}$ mod 227)

Man berechnet: $u = a^{-1} = 91$; $\lceil\sqrt{n}\rceil = 16$

$v = a^{\lceil\sqrt{n}\rceil} = 5^{16} = 175$

Dies ergibt die Liste

$$\mathcal{L}_1 = \{(0,1), (1,175), (2,207), (3,132), (4,173), (5,84), (6,172), (7,136), \underline{(8,192)}, (9,4), (10,19), (11,147), (12,74), (13,11), (14,109), (15,7)\}$$

Man durchläuft nun die Liste $\mathcal{L}_2$ und stellt folgende Fragen an Liste $\mathcal{L}_1$:

Treffer!

$y \cdot u^{x_2} = 86, 108, 67, 195, 39, 144, 165, 33, 52, 192,$

dazu $x_2 = 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9$

Ergibt $x_1 = 8$ und $x_2 = 9$, also $x = 8 \cdot 16 + 9 = \underline{\underline{137}}$