Faktorisierung

- algorithmisch schweres Problem (RSA basiert darauf.)
- für Zahlen von bestimmter Form: Angriffsmethoden bekannt

Beispiele

	geeignet für ...
• $(p-1)$-Methode	Ein Primfaktor p von n hat die Eigenschaft: $p-1$ hat nur kleine Primfaktor-Potenzen
• Pollard ρ-Methode	Ein Primfaktor p von n ist klein

"klein" bedeutet: $\leq B$

$\uparrow$

festgesetzte Schranke

$(p-1)$-Methode: Vorüberlegungen

Beobachtung I: Für jedes Vielfache k von $p-1$ gilt:

$$k = s \cdot (p-1) \quad \text{(für eine ganze Zahl } s\text{)}$$

$$\Rightarrow \quad a^k = a^{s(p-1)} = (a^{p-1})^s = 1^s = 1 \pmod p$$

$$\Rightarrow \quad a^k - 1 = 0 \pmod p \quad \Rightarrow \quad p \text{ ist Teiler von } a^k - 1$$

$$\Rightarrow \quad \mathrm{ggT}(a^k - 1, n) \geq p$$

Beobachtung II: Falls Primfaktor-Potenzen von $p-1$ alle $\leq B$ sind:

$$\Rightarrow \quad k := \prod_{\substack{q \text{ prim} \\ q^e \leq B}} q^e \quad \text{beinhaltet alle Primfaktor-Potenzen von } p-1$$

$$\Rightarrow \quad k \text{ ist Vielfaches von } p-1$$

$$\Rightarrow \quad \mathrm{ggT}(a^k - 1, n) \geq p$$

- **Test:** $1 < \mathrm{ggT}(a^k - 1, n) < n$
- Falls Test erfolgreich: $\mathrm{ggT}(a^k - 1, n)$ liefert Faktor von n.

$(p-1)$-Methode

FINDEFAKTOR(n, B)

Berechne $k := \prod_{\substack{q \text{ prim}, \\ q^e \leq B}} q^e$.

while (noch kein Faktor gefunden) **do**

1. Wähle zufällig eine Basis $a \in \mathbb{Z}_n^*$.
2. $f := \mathrm{ggT}(a^k - 1, n)$
3. **if** $(1 < f < n)$ **return** f // else: mache weiter

end

end

Aufgabe: Faktorisiere $n = 1'241'143$ durch Anwendung der $(p-1)$-Methode mit $B = 13$ und $a = 2$.

- Laufzeit der $(p-1)$-Methode: $\approx O(\mathrm{B})$
- **Begriff** B-Potenz-glatt bedeutet: alle Primfaktor-Potenzen sind $\leq B$ (englisch: powersmooth)

Pollard ρ-Methode: Grund-Idee

- geeignet für Zahlen, die einen kleinen Primfaktor p enthalten
- **Grund-Idee:** gesucht sind ganze Zahlen x, y mit der Eigenschaft:
  - $x \neq y \pmod{n}$ und
  - $x = y$ modulo einem Primfaktor p von n
- Dann: p teilt $x - y$ $\Rightarrow \mathrm{ggT}(x - y, n)$ liefert Faktor von n

Aufbau des Algorithmus

while (noch kein Faktor gefunden) **do**
1. Wähle zufällige Zahlen x, y $\leftarrow$ nach einem best. Muster
2. Berechne $f := \mathrm{ggT}(x - y, n)$
3. **if** $(1 < f < n)$ then return f // else: mache weiter

end

Geburtstagsparadoxon
bekanntes Ergebnis:

- bei 23 Personen haben mit Wahrscheinlichkeit > 0.5 zwei am gleichen Tag Geburtstag
- bei 30 Personen beträgt die entsprechende W'keit ≈ 0.7

Aufgabe: Wie gross ist die W'keit, dass unter 7 ausgewählten Personen keine zwei am gleichen Tag Geburtstag haben? (unter Vernachlässigung des 29. Februars)

Lösung: $p = \dfrac{365 \cdot 364 \cdot 363 \cdot 362 \cdot 361 \cdot 360 \cdot 359}{365^7} \approx 0.94$

Bem:
$$\begin{aligned} p &= \frac{365}{365} \cdot \frac{364}{365} \cdot \frac{363}{365} \cdots \frac{359}{365} \\ &= 1 \cdot \left(1 - \frac{1}{365}\right) \cdot \left(1 - \frac{2}{365}\right) \cdots \left(1 - \frac{6}{365}\right) \end{aligned}$$

Geburtstagsparadoxon – Verallgemeinerungen

W'keit für: keine zwei Personen haben am gleichen Tag Geburtstag

bisher: $p_7 = 1 \cdot \left(1-\frac{1}{365}\right) \cdot \left(1-\frac{2}{365}\right) \cdots \left(1-\frac{6}{365}\right)$

k Personen: $p_k = 1 \cdot \left(1-\frac{1}{365}\right) \cdot \left(1-\frac{2}{365}\right) \cdots \left(1-\frac{k-1}{365}\right)$

$365 \rightsquigarrow m$: $p_{m,k} = 1 \cdot \left(1-\frac{1}{m}\right) \cdot \left(1-\frac{2}{m}\right) \cdots \left(1-\frac{k-1}{m}\right)$

$\uparrow$

beim Ziehen von k Elementen aus m

Umformungen (unter Verwendung von $1-x \approx e^{-x}$) ergeben:

Resultat 1: $p_{m,k} \approx e^{\frac{-k(k-1)}{2m}}$

Resultat 2: Wenn $k > 1.2\sqrt{m}$ ist, dann gilt: $p_{m,k} \leq 0.5$.
resp.: Zieht man k Elemente aus einer Menge von m Elementen, so sind mit W'keit ≥ 0.5 zwei gleiche dabei. (Ziehen mit Zurücklegen)

Pollard ρ-Methode: Vorüberlegungen II

Erinnerung: Grund-Idee der Methode:

while (noch kein Faktor gefunden) **do**

1. Wähle zufällige Zahlen x, y $\leftarrow$ nach einem best. Muster
2. Berechne $f := \mathrm{ggT}(x - y, n)$
3. **if** $(1 < f < n)$ then return f // else: mache weiter

end

Aufgabe: $n := 143$. Ziehe 4 zufällige Zahlen aus $\mathbb{Z}_n$ und prüfe, ob darunter ein Paar x_i , x_j ist mit $1 < \mathrm{ggT}(x_i - x_j, n) < n$. Falls ja, bestimme daraus einen Faktor von n.
(Z.B. via `www.zufallsgenerator.net`.)

Fragestellung: Wie muss man (allgemein) die Anzahl Zufallszahlen k wählen, damit (wahrscheinlich) ein solches Paar dabei ist?

- "Gleichheit" bedeutet hier: "$= \pmod{p}$" für einen Faktor p von n
  $\Rightarrow m = p$
- Bed. von vorhin ergibt: Gewünschtes Paar ist mit W'keit ≥ 0.5 dabei, sofern $k > 1.2\sqrt{p}$ gilt.

Formulierung des Algorithmus: 1. Ansatz

while (noch kein Faktor gefunden) **do**
  Wähle eine zufällige Zahlen $x \in \mathbb{Z}_n$
  for alle bisher gezogenen Zahlen y **do**
    Berechne $f := \mathrm{ggT}(x - y, n)$
    if $(1 < f < n)$ then return f // else: mache weiter
  end
end

Analyse

- Betrachte ”kleinsten” Faktor p von n. (1 zählt hier nicht als Faktor.)
- Gemäss Bed. von letzter Folie:
  Nach $1.2\sqrt{p}$ Iterationen wird ein Faktor gefunden (mit W’keit $\geq 50\%$)
- Wenn p klein ist, dann ist der obige Algorithmus schnell.
- Analyse für Worst Case: $p \approx \sqrt{n}$
  $\Rightarrow 1.2\sqrt{\sqrt{n}} = 1.2n^{1/4}$ Iterationen nötig (im Schnitt)
- Verbesserungsmöglichkeit bez. Laufzeit: for-Schleife verkürzen!

Pollard ρ-Methode: Verbesserungsidee

Idee: Anstatt zufällige $x \in \mathbb{Z}_n$ zu ziehen:

- Nehme Funktion f, die sich "wie ein Zufallsgenerator" verhält, und
- bestimme damit fortlaufend die x-Werte.

Beispiel: $f(x) = x^2 + a \qquad$ mit $a \neq 0, -2$

Vorgehen:

- Wähle zufälliges x_0, und berechne rekursiv:

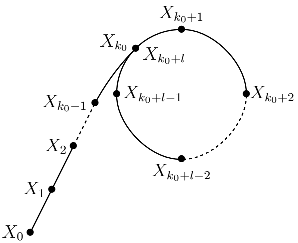
$$\begin{aligned} x_1 &= f(x_0) \\ x_2 &= f(x_1) \\ x_3 &= f(x_2) \\ etc \end{aligned}$$

- Rechne in allen Schritten jeweils modulo.

Aufgabe: $f(x) = x^2 + 1 \pmod{11}, \quad x_0 = 3$
Bestimme die ersten 10 Glieder.

Pollard ρ-Methode: Verbesserungsidee

Lösung: $$\underbrace{\overset{\overset{0}{\downarrow}}{3},10,2,5}_{\text{Vorperiode}},\ \underbrace{\overset{\overset{m}{\overset{k_0}{\downarrow}}}{4},6}_{\overset{\longleftrightarrow}{\ell}},\ \underbrace{\overset{\overset{n_0}{\downarrow}}{4},6},\underbrace{\overset{\overset{2m}{\downarrow}}{4},6},\ldots$$



Bezeichnungen

- ℓ: Länge der Periode.
- n_0: Erste Stelle, an der ein Wert zum zweiten Mal vorkommt.
- k_0: Stelle, an der dieser Wert zum ersten Mal vorkommt.
- m: Stelle in der ersten Periode **und** Vielfaches von ℓ. $(m \leq n_0)$

Bemerkungen

- $x_m = x_{2m}$
- "Erwartung": $n_0 \approx 1.2\sqrt{p}$ (s. vorherige Analyse)
  [Annahme: f verhält sich wie ein Zufallsgenerator.]
- Verbesserung für Algorithmus: vergleiche jeweils x_i und x_{2i}
  $\Rightarrow$ findet gewünschtes Paar nach $2m \leq 2n_0 \approx O(\sqrt{p})$ Schritten

Pollard ρ-Methode

1. Wähle $x_0 = y_0$ und a $(\neq 0, -2)$ zufällig, setze $i := 1$.
2. **while** (noch keine Kollision gefunden)
   2.1 Bestimme $x_i := (x_{i-1})^2 + a \pmod{n}$,
   und $y_i := \left((y_{i-1})^2 + a\right)^2 + a \pmod{n}$
   2.2 $d := \mathrm{ggT}(x_i - y_i, n)$
   2.3 **if** $(1 < d < n)$ **return** d
   2.4 $i := i + 1$

end

- Gemäss Def: $x_i := f(x_{i-1}) = x_{i-1}^2 + a \pmod{n}$
- Einsetzen ergibt:
  $y_i = x_{2i} = x_{2i-1}^2 + a = (x_{2i-2}^2 + a)^2 + a = (y_{i-1}^2 + a)^2 + a \pmod{n}$

Beispiel

Aufgabe: Bestimme mit der Pollard ρ-Methode einen Faktor von $n = 143$. Wähle dazu $x_0 = y_0 = 1$ und $a = 3$.