technische universität dortmund

Faktorisierungsalgorithmen

Moritz Kerger

05.12.2023

Faktorisierung

- Anwendungsfälle
- Komplexität/Laufzeit
- Probleme

Algorithmen

- Probedivision
- Fermat Faktorisierung
- Faktorisierung nach Lehmann
- Pollard-Rho

Was ist Faktorisierung?

Unter Faktorisierung versteht man allgemein das Zerlegen einer Zahl in nichttriviale Teiler. Jede natürliche Zahl kann als Produkt von Primzahlen dargestellt werden. Gibt es nur einen Primfaktor, so muss die Zahl selbst eine Primzahl sein.

Anwendungsfälle

- Faktorisierung algebraischer Terme.
- Linearfaktorzerlegung von Polynomen.
- Faktorisierung von Matrizen zur Lösung von LGS.
- Faktorisierungsalgorithmen können verwendet werden, um die Kryptografische Sicherheit des RSA-Verfahrens zu brechen.

Faktorisierung des RSA - Moduls

PublicKey(e, N), PrivateKey(d)

Ein Außenstehender kennt die zusammengesetzte Zahl N und e. Angenommen es existiert ein Faktorisierungsalgorithmus, der N in kurzer Zeit faktorisieren kann. Dann kann mit den Faktoren p, q folgendermaßen auf den PrivateKey geschlossen werden:

- Berechne $\Phi(N) = (p-1) \cdot (q-1)$
- d ist das multiplikative Inverse von e mod $\Phi(N)$.
- Suche ein $e \cdot d \equiv 1 \mod \Phi(N)$

Bei der Fermat Faktorisierung wird das nochmal an einem Beispiel demonstriert.

Komplexität von Faktorisierungsalgorithmen

Das Faktorisierungsproblem kann nach heutigem Wissensstand nur in exponentieller Laufzeit gelöst werden*. Für Zahlen mit vielen Stellen faktorisiert das Zahlkörpersieb am schnellsten.

$$O\left(\sqrt[3]{\frac{64}{9}}\sqrt[3]{\mathit{log}\ N}\sqrt[3]{(\mathit{log}\ \mathit{log}\ N)^2}\right)$$

* Ausnahme bildet Shor's Algorithmus

Komplexität von Faktorisierungsalgorithmen

technische universität dortmund

Große Primzahlen berechnen und multiplizieren ist einfacher, als die Faktoren eines großen Primzahlprodukts zu finden. Diese Komplexitätsdifferenz macht sich das RSA Verfahren zunutze.

Komplexität von Faktorisierungsalgorithmen

Rechenaufwand zur Faktorisierung bestimmter RSA-Zahlen:

- RSA-330 - 1991 - Mehrere Tage Rechenaufwand
- RSA-640 - 2005 - 5 Monate auf 80 AMD Opteron CPUs
- RSA-829 - 2020 - 2700 CPU-Jahre auf Intel Xeon CPUs
- RSA-2048 - ?

RSA-2048 ist mit bekannten Algorithmen auf einer gewöhnlichen Rechnerarchitektur nicht faktorisierbar.

Probedivision

Die Zahl N soll in ein Produkt von Primzahlen zerteilt werden.

$$N = \prod_{p \in \mathbb{P}} p^{n_i} \qquad\qquad n_i \in \mathbb{N}, N \in \mathbb{N}$$

- Generiere Liste von Primzahlen.
- Teile N so lange durch eine Primzahl, bis diese N nicht mehr teilt.
- Fahre mit der nächsten Primzahl fort, bis $p_i > \sqrt{N}$.

Probedivision - Beispiel 11 bits

Faktorisiere 1980

Probedivision - Beispiel 11 bits

Faktorisiere 1980

- $N := 1980/2 = 990$

Probedivision - Beispiel 11 bits

Faktorisiere 1980

- $N := 1980/2 = 990$
- $N := 990/2 = 495$

Probedivision - Beispiel 11 bits

technische universität dortmund

Faktorisiere 1980

- $N := 1980/2 = 990$
- $N := 990/2 = 495$
- $N := 495/3 = 165$

Probedivision - Beispiel 11 bits

Faktorisiere 1980

- $N := 1980/2 = 990$
- $N := 990/2 = 495$
- $N := 495/3 = 165$
- $N := 165/3 = 55$

Probedivision - Beispiel 11 bits

Faktorisiere 1980

- $N := 1980/2 = 990$
- $N := 990/2 = 495$
- $N := 495/3 = 165$
- $N := 165/3 = 55$
- $N := 55/5 = 11$

Probedivision - Beispiel 11 bits

Faktorisiere 1980

- $N := 1980/2 = 990$
- $N := 990/2 = 495$
- $N := 495/3 = 165$
- $N := 165/3 = 55$
- $N := 55/5 = 11$
- $5 > \sqrt{11}$, also brich ab.

Die Primfaktoren sind $\{2, 2, 3, 3, 5, 11\}$, also $2^2 \cdot 3^2 \cdot 5 \cdot 11 = N$

Probedivision - Beispiel 60 bits

Faktorisiere 999999866000004473

- $N := 999999866000004473/2$
- ...

Probedivision - Beispiel 60 bits

Faktorisiere 999999866000004473

- $N := 999999866000004473/2$
- ...
- $N := 999999866000004473/999999929 = 999999937$
- $999999929 > \sqrt{999999937}$, also brich ab.

Die Primfaktoren sind $\{999999929, 999999937\}$, also $999999929 \cdot 999999937 = N$

Probedivision - Limitierung

Für Eingaben mit großen Primfaktoren dauert das Verfahren lange. Im zweiten Beispiel sind zwei große Primfaktoren enthalten. Dafür braucht der Computer mehrere Sekunden (≈ 12). Typische RSA Schlüssellängen sind heutzutage 2048-4096 bits.

Laufzeit: $O(\sqrt{N})$

Laufzeit mit Primzahl-LUT $O\left(\frac{\sqrt{N}}{\log(N)}\right)$

Fermats Methode

Die Methode von Fermat nutzt die dritte Binomische Formel.
Ziel ist es, a und b zu finden, für die gilt:

$$N = a^2 - b^2$$

Dadurch ergeben sich dann die Faktoren

$$N = (a + b) \cdot (a - b) = p \cdot q$$

$$p = (a + b)$$

$$q = (a - b)$$

Fermats Methode

Wähle a ungefähr in der "Mitte" von N und schau dann, ob es ein b gibt, das den gleichen Abstand zu beiden Faktoren hat. Ist b keine solche Zahl, dann vergrößern wir a. Da der Algorithmus am Anfang von einem kleinen b und einem $a \approx \sqrt{N}$ ausgeht, funktioniert dieser Algorithmus besonders gut für ähnlich große Primfaktoren.

Fermats Methode - Algorithmus

Wir berechnen p und q folgendermaßen:

```
fermat_factor(N):
    a ← ⌈√N⌉ + 1
    while True:
        b ← √(a² − N)
        if isInteger(b):
            break
        a ← a + 1
    return (a+b), (a-b)
```

Fermats Methode - Beispiel

N $= 9998962920073 58227$

- Schritt 1:
  - $a = 999948145$
  - $b = \sqrt{a^2 - N} = 26107.140747312795$

Fermats Methode - Beispiel

N $= 999896292007358227$

- Schritt 1:
  - $a = 999948145$
  - $b = \sqrt{a^2 - N} = 26107.140747312795$
- Schritt 2:
  - $a = 999948146$
  - $b = 51783$
  - b ist ganzzahlig, also brich ab.

Fermats Methode - Beispiel

N = 999896292007358227

- Schritt 1:
  - $a = 999948145$
  - $b = \sqrt{a^2 - N} = 26107.140747312795$
- Schritt 2:
  - $a = 999948146$
  - $b = 51783$
  - b ist ganzzahlig, also brich ab.

p = (999948146 - 51783)
q = (999948146 + 51783)

Test: 999948146^2 - 51783^2 $= 999896292007358227$ ✓

Faktorisierung nach Lehmann

Die Methode von Lehmann nutzt unter anderem die Probedivision.

- Führe Probedivision bis $\sqrt[3]{N}$ durch.
- Wenn keine Faktoren gefunden wurden, besteht die Zahl aus zwei Primfaktoren, oder ist selbst Primzahl.
- Führe den nachfolgenden Algorithmus aus.

Faktorisierung nach Lehmann - Algorithmus


```
lehman_factor(N):
    for k ← 1..⌈∛N⌉ :
        for a ← ⌈2√(kN)⌉..⌊2√(kN) + ⁶√N/(4√k)⌋ :
            b ← √(a² − 4kN)
            if isInteger(b):
                return ggT(a + b, N)
```

$$\text{for } k \leftarrow 1..\left\lceil \sqrt[3]{N} \right\rceil, \quad \text{for } a \leftarrow \left\lceil 2\sqrt{kN} \right\rceil .. \left\lfloor 2\sqrt{kN} + \frac{\sqrt[6]{N}}{4\sqrt{k}} \right\rfloor, \quad b \leftarrow \sqrt{a^2 - 4kN}$$

Faktorisierung nach Lehmann - Beispiel

N = 999896292007358227

- Schritt 1:
  - k = 1
  - x = 1999896290
  - y = $\sqrt{x^2 - 4kn}$ = 52214,2

Faktorisierung nach Lehmann - Beispiel

N $= 999896292007358227$

- Schritt 1:
  - k $= 1$
  - x $= 1999896290$
  - y $= \sqrt{x^2 - 4kn} = 52214{,}2$
- Schritt 2:
  - x $= 1999896291$
  - $y = 82012, 9$

Faktorisierung nach Lehmann - Beispiel

technische universität dortmund

N = 999896292007358227

- Schritt 1:
  - k = 1
  - x = 1999896290
  - y = $\sqrt{x^2 - 4kn}$ = 52214,2
- Schritt 2:
  - x = 1999896291
  - $y = 82012, 9$
- Schritt 3:
  - x = 1999896292
  - y = 10725916356
  - $y = 103566$
  - gcd(1999896292+103566, N) = 999999929

Faktorisierung nach Lehmann - Laufzeit

Der Algorithmus terminiert für ungerade Zahlen.
Die Laufzeit für den ersten Schritt beträgt $\sqrt[3]{N}$. Werden in diesem keine Faktoren gefunden, beträgt die Laufzeit $\sqrt[3]{N} \cdot \log \log n$.

Für Beispiel 2 aus der Probedivision kann die Rechenzeit von 12 Sekunden auf 0.0002 Sekunden reduziert werden!

Pollard-Rho

Definiere Funktion $g(x) := x^2 - c \mod k$.
Für den Anfang ist $c = 1$ und $k = N$.

Idee: Wende die Funktion $g(x)$ immer wieder auf das Ergebnis an.

$$x_i = g^i(x_0)$$

Irgendwann wird wegen $\mod k$ ein Zyklus entstehen.

Pollard-Rho Zyklus

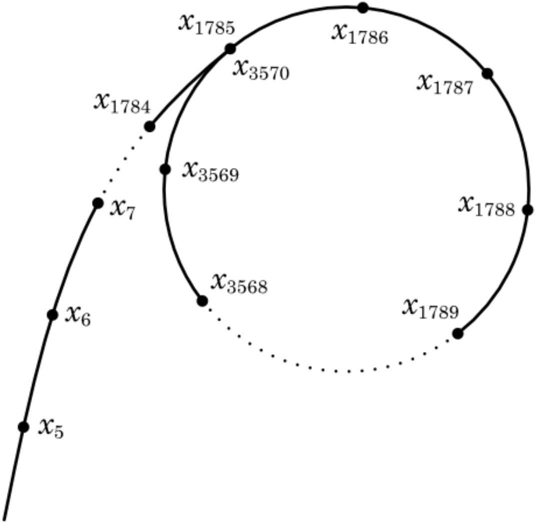

Figure: Charakteristischer Zyklus

Pollard-Rho - Algorithmus


```
pollard_rho(N):
    t,h ← 2, 2
    while True:
        t = g(t)
        h = g(g(h))
        d = gcd(|t-h|, N)
        if d == N:
            return False
        if d > 1:
            return False
    if d:
        return int(d)
```

Pollard-Rho - Algorithmus

technische universität dortmund

```
pollard_rho(N):
    t,h ← 2, 2
    while True:
        c ← 1
        while True:
            t = g(t)
            h = g(g(h))
            d = gcd(|t-h|, N)
            if d == N:
                d ← False
            if d > 1:
                break
        if d:
            return int(d)
        c ← c + 1
```

Pollard-Rho - Beispiel

technische universität dortmund

N $= 12999999077$

- Schritt 1:
  - t $= 2^2 - 1 = 3$
  - h $= (2^2 - 1)^2 - 1 = 8$
  - d $= gcd(5, N) = 1$

Pollard-Rho - Beispiel

$N = 12999999077$

- Schritt 1:
  - $t = 2^2 - 1 = 3$
  - $h = (2^2 - 1)^2 - 1 = 8$
  - $d = gcd(5, N) = 1$
- Schritt 2:
  - $t = 3^2 - 1 = 8$
  - $h = (8^2 - 1)^2 - 1 = 3968$
  - $d = gcd(3960, N) = 1$

Pollard-Rho - Beispiel

technische universität dortmund

N = 12999999077

- Schritt 1:
  - $t = 2^2 - 1 = 3$
  - $h = (2^2 - 1)^2 - 1 = 8$
  - $d = gcd(5, N) = 1$
- Schritt 2:
  - $t = 3^2 - 1 = 8$
  - $h = (8^2 - 1)^2 - 1 = 3968$
  - $d = gcd(3960, N) = 1$
- Schritt 3:
  - $t = 8^2 - 1 = 63$
  - $h = (3968^2 - 1)^2 - 1 = 8766871215$
  - $d = gcd(8766871152, N) = 13$
  - $d > 1$, also ist d Faktor von N
  - $p = 13$, $q = N/13 = 999999929$

Test: $13 \cdot 999999929 = 12999999077$ ✓

Pollard-Rho - Probleme

technische universität dortmund

- Für Primzahlen kein Abbruch

$$\texttt{gcd(|t-h|, N)} = 1 \text{ für } N \in \mathbb{P}$$

- c = 1 und k < N liefert das eventuell keine Faktoren.
- Für mehr als 2 Faktoren rekursive Anwendung auf das Ergebnis nötig.
- Bei 2 Primzahlen gleicher Länge terminiert der Algorithmus im Mittel deutlich später.

Pollard-Rho - Laufzeit

Erwartungswert der Zyklenlänge bei Zahl N:

$$\sqrt{k}$$

Da die Laufzeit von der Zyklenlänge und damit k abhängt, ist dieser Algorithmus besonders effizient für Zahlen, bei denen ein Faktor sehr klein ist. Handelt es sich beispielsweise um eine zusammengesetzte Zahl mit zwei ähnlich großen Primfaktoren, so terminiert der Algorithmus in der Hälfte aller Fälle nach

$$O\left(N^{\frac{1}{4}} \cdot \mathrm{Li}(N)\right)$$

Literaturverzeichnis

technische universität
dortmund

Recognizing Primes and Composites, pages 117–171.
Springer New York, New York, NY, 2005.

W. Y. Feng.
How to quickly factor a number: Pollard's rho algorithm.

B. R. S. Lehman.
Factoring large integers.
Mathematics of Computation, 28:637–646, 1974.

C. Pomerance and P. Erdös.
A tale of two sieves.
1998.

M. Pound and S. Riley.
Breaking rsa.

[1] [2] [3] [4] [5]