Der Miller-Rabin Primzahltest

Zusammenfassung

Für viele moderne Computeranwendungen, etwa für die Kryptographie, benötigt man zufällig gewählte Primzahlen bestimmter Größe, meist zwischen 128 und 4096 Bit. Bei Zahlen dieser Größenordnungen kann man jedoch nicht mehr die Schulmethode zur Überprüfung der Primalität einer gegebenen Zahl anwenden; Es bedarf hier einiger mathematischer Tricks und Transformationen, mit deren Hilfe man Primzahlen von zusammengesetzten Zahlen unterscheiden kann. Im Folgenden wird der Miller-Rabin Primzahltest vorgestellt, ein probabilistisches Verfahren, mit dessen Hilfe man mit einer voraussagbaren Wahrscheinlichkeit einen Primzahlkandidaten erkennen kann. Der Miller-Rabin Primzahltest wird folgend Stück für Stück entwickelt und anhand ausgesuchter Beispiele genau erläutert. Abschließend wird eine Analyse des Algorithmus folgen, deren Hauptaugenmerk auf der Fehlerwahrscheinlichkeit liegt.[1]

[1] Grundlage des vorliegenden Dokuments ist das Buch von T.H. Cormen, C.E. Leiserson und R.L. Rivest: „Introduction to Algorithms“, MIT Press/McGraw-Hill (1990). Diese Arbeit versteht sich nicht als formale (rein wissenschaftliche) Darstellung der Thematik, sondern richtet sich an interessierte „Nicht-Mathematiker“.

1. Einführende Betrachtungen

Für viele moderne Computeranwendungen, etwa für die Kryptographie, benötigt man zufällig gewählte Primzahlen bestimmter Größe (meist zwischen 128 und 4096 Bit). Glücklicherweise sind große Primzahlen nicht selten, so dass es nicht all zu viel Zeit kostet, aus einer Menge zufällig gewählter Zahlen, einen Primkandidaten zu finden. Für die Frage der Häufigkeit von Primzahlen gibt es die Funktion ***π(n)***, welche die Anzahl der Primzahlen $\leq n$ angibt. So gilt z.B. für ***π(10) = 4***. Wie wir schon aus der Schule wissen, sind dies die Zahlen *2, 3, 5* und *7*.

Der Gauß'sche[2] Primzahlsatz liefert uns eine statistische Aussage über einen Bereich in dem Primzahlen liegen können, es gilt:

$$\lim_{n\to\infty} \frac{\pi(n)}{n/\ln(n)} = 1, \text{ so dass gilt } 1*\pi(n) = \frac{n}{\ln(n)}$$

Die Formel lässt sich auf die einfache Aussage zurückführen, dass der Quotient *n/ln(n)* asymptotisch äquivalent zu ***π(n)*** ist dies ist die Kernaussage des Primzahlsatzes. Mit dem Beweis dieser Approximation stellt sich ein weiteres Problem, welches erst am Ende des neunzehnten Jahrhunderts gelöst wurde.

Im Jahre 1848 zeigte Tschebyschow (1821-1894), dass der Grenzwert

$$\lim_{n\to\infty} \frac{\pi(n)}{n/\ln(n)}$$

falls dieser existiert, den Wert *1* hat, was den Beweis des Primzahlsatzes bedeuten würde. Tschebyschow konnte diesen leider nicht finden. Erst 1896 bewiesen die Mathematiker Hadamard (1865-1963) und De La Vallée Poussin (1866-1962) unabhängig voneinander den Primzahlsatz.

Wir können diese Approximation nutzen, um zu bestimmen, wie viele Zahlen wir in einer Umgebung von *n* zufällig wählen müssen, so dass eine davon prim ist. Möchten wir z.B. eine 512-Bit lange Zahl finden, so müssen wir in etwa $ln(2^{512}) = 354{,}8913$, *also ~355* Zahlen in der Umgebung von *n* bestimmen, um eine Primzahl zu finden.

Nachdem wir nun wissen, dass Primzahlen nicht all zu selten sind, müssen wir uns nun ein Verfahren überlegen, mit dem man feststellen kann, ob eine zufällig gewählte Zahl tatsächlich prim ist. Eine einfache Methode ist die so genannte Probedivision *(engl. trial-testing)*. Die Grundidee dieses Verfahrens ist, *n* sukzessive durch Primzahlen *2, 3, 5, 7, 11, ...,* $\leq\sqrt{n}$ zu teilen. Wenn es keine Teiler gibt, so ist die Zahl prim, wenn es Teiler gibt, so ist sie nicht prim, in diesem Falle liefert uns dieses Verfahren dann auch gleich die sog. Primfaktoren (d.h. die Primteiler) der zusammengesetzten Zahl. Die Faktorisierung von zusammengesetzten Zahlen soll uns an dieser Stelle allerdings nicht weiter beschäftigen. Weitere Informationen hierzu findet man z.B. in dem eingangs genannten Buch von Cormen, Leiserson und Rivest.

Die Probedivision bietet den Vorteil, dass sie auf jeden Fall beweist, ob eine Zahl prim ist oder nicht. Leider ist dieses Verfahren nur bei relativ kleinen Zahlen effizient zu gebrauchen, da der Rechenaufwand exponentiell steigt. Ist *n* binär kodiert, so ergibt sich für eine Bitlänge von δ ein Aufwand von $O(\sqrt{n}) \approx 2^{(\delta/2)}$.

[2] C. Friedrich Gauß (1777-1855).

Nehmen wir an, dass eine gegebene Zahl 58 Bit benötigt, dann müssen im Falle einer Primzahl etwa 536.870.912 Tests durchgeführt werden, bis die Probedivision ein Ergebnis liefert. Nehmen wir an, dass ein Rechner in einer Sekunden *1000* Überprüfungen durchführen kann, dann bräuchte er in etwa 6 Tage, um mittels der Probedivision eindeutig zu bestätigen, dass diese Zahl prim ist. Dies macht das Verfahren praktisch nutzlos, da das *trial-testing* bei großen Zahlen, wie man sie heute bei vielen Computerapplikationen benötigt, wegen der langen Berechnungszeit nicht einsetzbar ist, müssen wir uns eine andere Methode für den Primzahltest ausdenken.

Der französische Jurist und Mathematiker Fermat hilft uns mit einem „kleinen“ aber wichtigen Satz:

$$a^{(n-1)} \equiv 1 \pmod{n},\ a \in [2..n-1]$$

nur dann, wenn *n* prim ist.

Beweis:

Schreiben wir „rein zufällig“ folgendes Produkt auf:

$$(1*a)*(2*a)*(3*a)*...*[(n-1)*a]$$

Dies kann man umformen, und zwar

$$(n-1)!* a^{(n-1)}, \text{ dies bedeutet, dass}$$

$$(n-1)! * a^{(n-1)} \equiv (n-1)! \pmod{n} \text{ gilt.}$$

Teilt man nun durch *(n-1)!*, so erhält man

$$a^{(n-1)} \equiv 1 \pmod{n}.$$

Was bringt uns der Satz von Fermat? Nun, wir müssen nur ein $a \in [2..n-1]$ finden, so dass $a^{(n-1)} \neq 1 \pmod{n}$ gilt und wir können ausschließen, dass *n* eine Primzahl ist. In diesem Fall spielt die Zahl *a* die Rolle eines Belastungszeugen (*engl. witness*) dafür, dass *n* eine zusammengesetzte Zahl ist. Natürlich rechnet man nicht alle $a \in [2..n-1]$ durch, da der Rechenaufwand ähnlich hoch wird, wie beim anfangs erwähnten *trial-testing*. Meistens nimmt man für $a = 2$ an, so dass sich folgende Programmfunktion ergeben könnte:

```
Funktion fermat_witness

Eingabe:  Zahl n

Ausgabe:  true falls ein Zeuge gefunden ist, false sonst

function fermat_witness(n: integer): boolean

begin

  if (2^(n-1) mod n = 1) then

    return false    // kann prim sein

  else

    return true     // ist auf keinen Fall prim

end
```

Liefert diese Funktion als Ergebnis *true*, so können wir absolut sicher sein, dass *n* **keine Primzahl** ist.[3] Liefert die Funktion jedoch *false* zurück, kann es sich unter Umständen um eine so genannte Pseudoprimzahl zur *Basis 2* handeln, dies sind zusammengesetzte Zahlen, die den Fermat-Test zur *Basis 2* „bestehen". Nun stellt sich natürlich die Frage, wie viele dieser Pseudoprimzahlen existieren.

Glücklicherweise sind dies sehr wenige. Nehmen wir z.B. alle Primzahlen $2 \leq p \leq 10.000$, so existieren im genannten Intervall gerade einmal 22 dieser Pseudoprimzahlen. Man kann beweisen, dass die Chance, zufällig gerade so eine Pseudoprimzahl zu treffen, gegen $\rightarrow 0$ geht, wenn *n* wächst.

Was wäre, wenn man andere Basen verwendet, so könnten wir statt Basis 2 die 3 nehmen, oder? Leider kommen wir da vom Regen in die Traufe, da auch Pseudoprimzahlen zur Basis 3, z.B. 1105[4] existieren. Es existieren sogar zusammengesetzte Zahlen, die diesen Test für mehrere Basen bestehen. Ferner gibt es zusammengesetzte Zahlen, die den Fermat-Test für alle Basen $b \in Z_n^*$ „unbeschadet" überstehen.[5]

Diese Zahlen nennt man Carmichael-Zahlen, benannt nach Robert D. Carmichael (1879-1967), der sie erstmals um 1910 beschrieb. Die ersten Carmichael Zahlen sind 561 (3x11x17), 1105, 1729, 2465, 2821... Dass dies stimmt, kann man mit einem guten Taschenrechner oder mit einer funktionalen Programmiersprache wie Haskell einfach durch nachrechnen prüfen. So liefert Haskell für folgende Eingabe

```
Main> ([x|x <-[2..560], x^(560) `mod` 561 == 1]) == ([x|x <-[2..560], ggt(x,561)== 1])
```

true. D.h., die Menge der Elemente von Z_{561}^* ist identisch mit der Menge der Zahlen aus Z_{561}^*, für die gilt $x^{(560)} \equiv 1 \ (mod\ 561)$.

---

[3] Der „Zeuge" 2 bestätigt uns sozusagen, dass *n* ein „falscher Fünfziger ist".

[4] Andere Beispiele: 341 = ist eine 2-PRP, 91 = ist eine 3-PRP, 217 = ist eine 5-PRP und 25 = ist eine 7-PRP.

[5] $Z_n^* := \{a \ Z_n : ggT(a, n) = 1\}$

Wie Basis-2-Pseudoprimzahlen sind Carmichael-Zahlen extrem selten, es gibt nur 255 aus 100.000.000, dies ist jedoch kein Grund, sie nicht weiter zu beachten. Wir müssen unseren `fermat_witness` Algorithmus demzufolge so „absichern“, dass er sich nicht hinters Licht führen lässt.

2. Der Miller-Rabin Primzahltest

Ziel ist im Folgenden, aus Fermats Kriterium ein stärkeres herzuleiten, **das nicht nur notwendig, sondern auch hinreichend** für die *Primalität* einer gegebenen Zahl *n* ist. Diese Verfeinerung führt zum Miller-Rabin Primzahltest[6]. Dieser Test basiert auf den Überlegungen von Miller, auf deren Grundlage Rabin in den 80'er Jahren des zwanzigsten Jahrhunderts den nach ihnen genannten Test entwickelte. Dieses Testverfahren löst die Probleme unserer ersten Funktion durch folgende Modifikationen:

i) Wir benutzen weiterhin den Fermat-Test, verwenden mehrere „echt zufällig“[7] gewählte Basen und nicht nur eine, außerdem muss *n* ungerade sein, da ohnehin nur ungerade Zahlen prim sein können.

ii) Eine Zahl *x* gilt ebenfalls als Belastungszeuge für *n*, wenn $x^2 \equiv 1 \pmod n$ ist, aber $x \neq 1$ und $x \neq -1 \leftrightarrow n-1$.[8]

Wir stellen uns zunächst die Frage, wie man den Fermat-Test und die Suche nach der nicht trivialen Quadratwurzel von *1* geschickt verbinden kann. Nehmen wir an, dass die zu prüfende Zahl $n = 561$ sei[9], damit ist unser $n-1 = 560$. Sehen wir uns zunächst die binäre Darstellung der Zahl *560* an:

```
512 | 256 | 128 | 064 | 032 | 016 | 008 | 004 | 002 | 001

  1     0     0     0     1     1     0     0     0     0 = 560 ( = u)
```

Betrachten wir die letzen vier in Fettdruck gehaltenen Nullen. Streichen wir diese vier Nullen und schieben die übrig bleibenden Stellen um *4 (=t)* nach rechts, dann erhalten wir

```
512 | 256 | 128 | 064 | 032 | 016 | 008 | 004 | 002 | 001

                          1     0     0     0     1     1 =  35
```

die Zahl *35*. Um wieder $(n-1) = 560$ zu erhalten, rechnen wir einfach

$$(2^4) * 35 = 16 * 35 = 560.$$

Was soll das? Nun, mit diesem kleinen Trick können wir den Fermat-Test und die Suche nach einer nicht-trivialen Quadratwurzel sehr geschickt kombinieren. Wir berechnen zunächst $a^{35} \pmod n$, dann quadrieren wir dieses Ergebnis *4-mal* und können während wir Quadrieren, feststellen, ob eine nicht-triviale Quadratwurzel der *1* existiert.

---

[6] Gary Miller entwickelte den Test um 1976, verwendete hierzu aber die damals noch unbewiesene Riemann-Hypothese. Michael Rabin gelang es dann 1980 auch dieses Manko zu beheben und stellte den Test auf eine sichere mathematische Basis.

[7] Hier sollte man ggf. kryptografisch starke PRNG verwenden, die üblichen PRNG heute eingesetzter Programmiersprachen sollten vor allem im Bereich der Kryptografie nicht verwendet werden.

[8] Wenn *n* eine Primzahl ist, dann ist Z_n ein Körper. In einem Körper hat jede quadratische Gleichung höchstens zwei verschiedene Lösungen (das werde ich hier aber nicht beweisen). Nun hat aber in Zn^* die quadratische Gleichung $x^2 \equiv 1 \pmod n$ schon die beiden Lösungen $x = 1$ und $x = -1 \leftrightarrow n-1$. Da $n > 2$ vorausgesetzt werden kann, sind diese auch verschieden. Gibt es nun eine weitere Lösung x, dann kann Z_n kein Körper sein. Ist Z_n kein Körper, folgt messerscharf, dass *n* auch keine Primzahl sein kann. Findet man ein $x \neq 1$ und $x \neq -1$, mit $x^2 \equiv 1 \pmod n$, spricht man von einer nicht-trivialen Quadratwurzel von $1 \bmod n$.

[9] also eine Carmichealzahl

Sei $a = 2$, dann sieht die Berechnung des Miller-Rabin Primzahltest für $n = 561$ in etwa so aus:

i) $2^{35} \pmod{561} = 34.359.738.368 \pmod{561} = 263$

ii) $263^2 \pmod{561} = 69.169 \pmod{561} = 166$

iii) $166^2 \pmod{561} = 27.556 \pmod{561} = 67$

iv) $67^2 \pmod{561} = 4.489 \pmod{561} = 1$

Wir haben soeben eine nicht-triviale Quadratwurzel gefunden, d.h. n ist keine Primzahl, was ja stimmt, da unser $n = 561$, wie wir wissen, eine Carmichael-Zahl, war.

Die folgende Funktion `witness` realisiert den Fermat-Test zur Basis a. Die Berechnung von $a^{(n-1)} \bmod n$ wird wie folgt realisiert:

i) Bestimme eine ungerade Zahl u und eine Zahl t, so dass gilt $(2^t)*u = (n-1)$. Dies funktioniert stets, da n laut Voraussetzung ungerade ist, muss $(n-1)$ gerade sein, daher lässt sich stets eine ungerade Zahl u und eine Zahl t finden, so dass $(2^t)*u = (n-1)$ gilt.

ii) Berechne nun $a^u \bmod n$, dann quadriere dieses Ergebnis t mal. Nach jedem Quadrieren des Zwischenergebnisses x wird jedoch zusätzlich geprüft, ob das Ergebnis 1 ist. War das zuvor berechnete $x \neq 1$ und $x \neq n-1$, so ist ein Belastungszeuge gefunden, durch den n als zusammengesetzt entlarvt wird, und die Funktion bricht mit dem Rückgabewert *true* ab.

iii) Zum Schluss wird wie im Fermat-Test geprüft, ob $a^{(n-1)} \bmod n \neq 1$ ist. Wenn ja, wird *true* zurückgegeben, d.h. „Belastungszeuge“ gefunden, sonst *false*.

Eine Realisierung als Programm könnte wie folgt aussehen:

```
Funktion witness
Eingabe:  Zahl a, Zahl n
Ausgabe:  true falls ein Zeuge dafür, dass n nicht prim, false
          sonst
function witness(a, n: integer): boolean
begin
  n-1 := (2^t)*u, wobei t ≥ 1 und u ungerade
  x0 := modular_exp(a,u,n);
  for i := 1 to t do
  begin
    xi := xi-1^2 mod n;
    if (xi = 1 and xi-1 <> 1 and xi-1 <> n-1) then return true;
  end
  if (xt <> 1) then
     return true    // a ist Belastungszeuge gegen Primalität
  else
     return false;  // a ist kein Belastungszeuge
end
```

Bei der Zahl $n = 561$ liefert der Fermat-Test mit dem Zeugen 2 nicht das Ergebnis, dass n zusammengesetzt ist. Die Funktion `witness` dagegen findet eine nicht-triviale Quadratwurzel, wie wir bereits oben gesehen haben. Damit lässt sich 561 als zusammengesetzt identifizieren.

Der Miller-Rabin Primzahltest besteht nun schlicht aus einem s-maligen Aufruf der Funktion `witness` mit zufällig gewählten Zeugen $a \in \{1, ..., n-1\}$.

```
Funktion is_miller_rabin_prime
Eingabe:  Zahl n, Anzahl der Versuche s
Ausgabe:  true falls Miller-Rabin prim, false sonst
function is_miller_rabin_prime(n, s: integer): boolean
begin
  for j := 1 to s do
  begin
    a := random(2,n-1);
    if (witness(a,n)) then return false; //100% nicht prim
  end
  return true; //[(½^s)*100]% „miller-rabin" prim
end
```

Der Miller-Rabin Algorithmus ein so genannter Monte-Carlo-Algorithmus. Monte Carlo-Algorithmen laufen schneller als deterministische Algorithmen, sie liefern aber mit einer gewissen Wahrscheinlichkeit ein falsches Ergebnis. Monte-Carlo Algorithmen gehören zur Klasse sog. probabilistischer Algorithmen, dies sind Algorithmen, in denen der Zufall eine wichtige Rolle spielt. In probabilistischen Algorithmen werden Zufallszahlen ermittelt, anhand derer eine Entscheidung gefällt wird, dadurch läuft ein solcher Algorithmus jedes Mal anders, auf diese Weise wird dann versucht eine Lösung zu finden. Deterministische Algorithmen brauchen dagegen oft sehr viel Zeit um Ergebnisse zu liefern, siehe beispielsweise *trial-divison*. Natürlich besteht die Gefahr, dass die Ergebnisse solcher Monte-Carlo Algorithmen nicht immer richtig sind, aus diesem Grunde sollte man stets bestimmen, wie hoch die Chance ist, ein falsches Ergebnis zu erhalten. Dies werden wir im nun folgenden Abschnitt auch machen.

3. Fehlerrate des Miller-Rabin Primzahltest

Im Gegensatz zum einfacheren Fermat-Test gibt es beim Miller-Rabin Primzahltest keinen schlechten Input in Form einer Carmicheal-Zahl oder *Pseudo-**X**-Primzahl*. Alles ist abhängig von der Anzahl der Tests und dem Glück bei der zufälligen Wahl der Basen *a*.

Um nun zu beweisen, dass die Anzahl der so genannten Entlastungszeugen $\leq (n-1)/2$ beträgt, werde ich kurz Lagrange`s Theorem angeben:

Wenn (S, x) eine endliche Gruppe ist und (S',x) eine Untergruppe von (S,x), dann

$|S'| \mid |S|$, d.h. $|S'|$ ist ein Divisor von $|S|$.

Eine Untergruppe S' heißt echte (nicht-triviale) Untergruppe, wenn $S' \neq S$ bzw. $S' \neq \{e\}$ gilt. Ist S' eine nicht-triviale Untergruppe einer endlichen Gruppe S, so gilt $|S'| \leq |S|/2$.

In Bezug auf Miller-Rabin heißt das, dass die Anzahl der Entlastungszeugen $\leq (n-1)/2$ ist. Das wiederum bedeutet, dass die Anzahl der Belastungszeugen mindestens $(n-1)/2$ ist. Wir müssen nun zeigen, dass es eine echte Untergruppe von Z_n^* gibt, die alle Nichtbelastungszeugen enthält.[10] Eine solche Untergruppe besitzt, wie wir wissen $\leq (n-1)/2$ Elemente. Um dies zu zeigen, teilen wir den Beweis in zwei Fälle auf:

1. Fall:

Es gibt ein $x \in Z_n^*$, so dass $x^{(n-1)} \neq 1 \pmod n$ gilt. Mit anderen Worten: *n* ist keine Carmichael-Zahl, dieser Fall tritt in der Praxis übrigens am Häufigsten auf. Wir definieren die Menge der Entlastungszeugen mit

$$E = \{b \in Z_n^* \mid b^{(n-1)} \equiv 1 \pmod n\}$$

B ist nicht leer, da $1 \in$ E. Da E unter der Multiplikation *mod n* abgeschlossen ist, können wir sagen, dass E eine Untergruppe von Z_n^* ist. Man beachte, dass jeder Entlastungszeuge $\in$ E ist, da *a* der folgenden Gleichung genügt $a^{(n-1)} \equiv 1 \pmod n$. Damit kann *x* nicht Element von E sein, da es aber $\in Z_n^*$ ist, gilt $x \in Z_n^* - E$, damit ist E eine echte/nicht-triviale Untergruppe von Z_n^*. Daraus folgt, dass in E maximal (n-1)/2 Elemente (Entlastungszeugen) enthalten sind.

2. Fall:

Für alle $x \in Z_n^*$ gilt $x^{(n-1)} \equiv 1 \pmod n$. Mit anderen Worten: *n* ist eine Carmichael-Zahl, dieser Fall ist äußerst selten, trotzdem ist der Miller-Rabin Primzahltest in der Lage, diesen zu „erkennen“, anders als die Testverfahren, die nur auf dem kleinen Satz von Fermat basieren.

Da *n* Carmichael-Zahl ist (damit keine Potenz einer Primzahl), können wir diese zusammengesetzte Zahl auch in der Form $n_1 n_2$ schreiben, wobei n_1 und n_2 ungerade, relativ prim zueinander und > 1 sind. Das könnte dann wie folgt aussehen:

---

[10] Z_n^* ist hierbei definiert als: $Z_n^* := \{a \, Z_n : ggT(a, n) = 1\}$

$$n = p_1^{e_1} * p_2^{e_2} * ... * p_r^{e_r},$$

damit ist

$$n_1 = p_1^{e_1} \text{ und } n_2 = p_2^{e_2} * ... * p_r^{e_r}.$$

Überlegen wir uns nun, wie unsere echte Untergruppe von Entlastungszeugen aussehen muss. Auf jeden Fall enthält B alle Elemente *x*, für die gilt $x^{(n-1)} \equiv 1 \pmod n$. Könnten da noch andere Elemente drin sein? JA.

Überlegen wir uns kurz, welche Zahlen bei unserem Miller-Rabin `witness`-Test auftreten können. Entweder sind nach Berechnung von $x^u \bmod n$ alle Zahlen $= 1$ (diese Zahlen sind bereits in *B* oder sie sind $x^{(n-1)} \neq 1$ und $x^{(n-1)} \neq -1 \leftrightarrow n-1$, dieser Fall interessiert uns hier aber nicht. Interessanter ist der Fall, dass an einer Stelle unserer Berechnungen $(n-1) \leftrightarrow -1$ herauskommt. Wieso? Na ja, *-1* wird, soweit wir noch mal quadrieren zu 1 und gehört daher auch zur Gruppe B. Es folgt also:

$$B = \{x \in Z_n^*: x^{[(2^i)*u]} \equiv +/- 1 \pmod n\} \text{ für } i \text{ aus } [0,...,t]$$

Wir zeigen nun, dass es ein *w* gibt, das folgendes erfüllt

$$w^{[(2^i)*u]} \equiv -1 \pmod{n_1}$$

$$w^{[(2^i)*u]} \equiv 1 \pmod{n_2}, \text{ wobei } i \text{ aus } [0,...,t] \text{ ist.}$$

Daraus folgt aber, dass $w^{[(2^i)*u]} \neq 1 \pmod{n_1}$ ist, sowie $w^{[(2^i)*u]} \neq -1 \pmod{n_2}$, dies impliziert, dass $w^{[(2^i)*u]} \neq +/- 1 \pmod n$.

Wie kommt man auf $w^{[(2^i)*u]} \neq +/- 1 \pmod n$?

Korollar (welches hier nicht bewiesen wird):

$$a \equiv z \pmod n, \text{ g.d.w.}$$

$$a \equiv z \pmod{n_1}, a \equiv z \pmod{n_2}, ..., a \equiv z \pmod{n_i}$$

wobei $n_1,..,n_i$ paarweise prim zueinander sind. Nachdem wir aber voraussetzten, dass

$w^{[(2^i)*u]} \equiv -1 \pmod{n_1}$ sowie $w^{[(2^i)*u]} \equiv 1 \pmod{n_2}$ wird die gerade aufgestellte Forderung nicht erfüllt. Daher kann dieses *w* nicht Element der Gruppe $B = \{x \in Z_n^*: x^{[(2^i)*u]} \equiv +/- 1 \pmod n\}$ für *i* aus *[0,...,t]}* sein. Daher ist B eine echte/nicht-triviale Untergruppe, so dass $|B| \leq (n-1)/2$.

Was bedeutet dies jetzt für den Miller-Rabin Primzahltest? Beim ersten Durchlauf ist die Chance ½ (das ½ kommt von der Gruppenordnung, die ja maximal *(n-1)/2* ist, also in etwa die Hälfte der Elemente von Z_n^*), dass wir fälschlicherweise einen Entlastungszeugen erwischen, der behauptet *n* sei prim, obwohl dies nicht der Fall ist. Testen wir erneut, ist die Chance wieder ½, nachdem wir aber schon zuvor getestet haben, ergibt sich Aufgrund der Produktwahrscheinlichkeit eine Chance von *1/4, 1/8, 1/16*, usw. Bei *s* Durchläufen kommt man daher auf $[(\frac{1}{2})^s]*100$ % Fehlerwahrscheinlichkeit.

Bereits für $s = 50$ erhalten wir eine Fehlerwahrscheinlichkeit von weniger als 8,881784197001252323389053344726e-14%. Es wäre schier ein Unding, wenn diese Zahl doch zusammengesetzt ist und der Miller-Rabin Primzahltest versagt hat aber es kann sein.

An dieser Stelle sei angemerkt, dass man die Fehlerwahrscheinlichkeit sogar auf $(1/4)^s$ abschätzen kann, dies würde allerdings den Rahmen dieser Arbeit sprengen. Der hier dargestellte Beweis folgt dem in von Cormen, Leiserson und Rivest: „Introduction to Algorithms“, MIT Press/McGraw-Hill (1990) dargestellten. Einen Beweis zur Abschätzung auf $(1/4)^s$ findet man z.B. in N. Blum, „Theoretische Informatik Eine anwendungsorientierte Einführung“, Oldenburg-Verlag, (2001).

4. Weitere Informationen

Stimmt das?

„Behauptung“: Wenn $a^{(n-1)} \equiv 1 \pmod{n}$, für a = 1, dann ist *n* eine Primzahl.

Interessante Webseiten

Unter dem URL www.jjam.de/Java/Applets/Primzahlen/Miller_Rabin.html findet der interessierte Leser weitere Informationen zur Thematik, außerdem kann man auf der Webseite mit einem sog. Java-Applet Zahlen mittels Miller-Rabin Primzahltest auf ihre *Primalität* testen.

Eine formale Darstellung findet man auf den Seiten der Wikipedia unter dem URL http://de.wikipedia.org/wiki/Miller-Rabin-Test.

Weiterführende Literatur

T.H. Cormen, C.E. Leiserson und R.L. Rivest: „Introduction to Algorithms“, MIT Press/McGraw-Hill (1990)

Song Y. Yan: „Number theory for computing“, Springer-Verlag, (2002)

N. Blum, „Theoretische Informatik“, Oldenburg-Verlag, (2001)

A. Beutelspacher, „Lineare Algebra“, Vieweg-Verlag, (2001)

Einige Carmichael-Zahlen[11]

561 3 11 17
1105 5 13 17
1729 7 13 19
2465 5 17 29
2821 7 13 31
6601 7 23 41
8911 7 19 67
10585 5 29 73
15841 7 31 73
29341 13 37 61
41041 7 11 13 41
46657 13 37 97
52633 7 73 103
62745 3 5 47 89
63973 7 13 19 37
75361 11 13 17 31
101101 7 11 13 101
115921 13 37 241
126217 7 13 19 73
162401 17 41 233
172081 7 13 31 61
188461 7 13 19 109
252601 41 61 101
278545 5 17 29 113
294409 37 73 109
314821 13 61 397
334153 19 43 409
340561 13 17 23 67
399001 31 61 211
410041 41 73 137
449065 5 19 29 163
488881 37 73 181
512461 31 61 271
530881 13 97 421
552721 13 17 41 61
656601 3 11 101 197
658801 11 13 17 271
670033 7 13 37 199
748657 7 13 19 433
825265 5 7 17 19 73
838201 7 13 61 151
852841 11 31 41 61
997633 7 13 19 577
1024651 19 199 271
1033669 7 13 37 307
1050985 5 13 19 23 37
1082809 7 13 73 163
1152271 43 127 211
1193221 31 61 631
1461241 37 73 541
1569457 17 19 43 113
1615681 23 199 353
1773289 7 19 67 199
1857241 31 181 331
1909001 41 101 461
2100901 11 31 61 101
2113921 19 31 37 97
2433601 17 37 53 73
2455921 13 19 61 163
2508013 53 79 599
2531845 5 19 29 919
2628073 7 37 73 139
2704801 11 29 61 139
3057601 43 211 337
3146221 13 31 37 211
3224065 5 13 193 257
3581761 29 113 1093
3664585 5 29 127 199
3828001 101 151 251
4335241 53 157 521
4463641 7 13 181 271
4767841 13 19 97 199
4903921 11 31 73 197
4909177 7 13 73 739
5031181 19 23 29 397

[11] Es sei darauf hingewiesen, dass es im Internet (man suche z.B. mit Google) weitaus größere Listen von Carmichael-Zahlen gibt.

5049001 31 271 601
5148001 41 241 521
5310721 13 37 61 181
5444489 29 197 953
5481451 31 151 1171
5632705 5 13 193 449
5968873 43 127 1093
6049681 11 31 113 157
6054985 5 53 73 313
6189121 61 241 421
6313681 11 17 19 1777
6733693 109 163 379
6840001 7 17 229 251
6868261 43 211 757
7207201 17 353 1201
7519441 41 241 761
7995169 7 13 103 853
8134561 37 109 2017
8341201 11 31 61 401
8355841 13 41 61 257
8719309 19 37 79 157
8719921 7 23 41 1321
8830801 7 19 67 991
8927101 31 37 43 181
9439201 61 271 571
9494101 23 61 67 101
9582145 5 23 97 859
9585541 7 31 163 271
9613297 19 29 73 239
9890881 7 11 13 41 241
10024561 71 271 521
10267951 67 331 463
10402561 13 29 41 673
10606681 31 43 73 109
10837321 11 31 61 521
10877581 11 13 29 43 61
11119105 5 17 257 509
11205601 11 17 31 1933
11921001 3 29 263 521
11972017 43 433 643
12261061 19 61 71 149
12262321 17 41 73 241
12490201 19 37 109 163
12945745 5 19 29 37 127
13187665 5 17 113 1373
13696033 13 17 29 2137
13992265 5 7 19 53 397
14469841 73 379 523
14676481 71 421 491
14913991 43 127 2731
15247621 61 181 1381
15403285 5 37 139 599
15829633 43 547 673
15888313 7 19 67 1783
16046641 13 37 73 457
16778881 7 17 19 41 181
17098369 113 337 449
17236801 151 211 541
17316001 53 157 2081
17586361 13 61 67 331
17812081 7 41 53 1171
18162001 11 13 17 31 241
18307381 29 61 79 131
18900973 7 13 229 907
19384289 89 353 617
19683001 13 37 151 271
20964961 17 19 47 1381
21584305 5 17 197 1289
22665505 5 17 97 2749
23382529 97 193 1249
25603201 13 29 113 601
26280073 73 157 2293
26474581 7 19 67 2971
26719701 3 29 197 1559
26921089 13 37 97 577
26932081 11 47 113 461
27062101 11 31 61 1301
27336673 7 13 23 37 353
27402481 31 43 61 337
28787185 5 7 19 73 593
29020321 11 37 113 631
29111881 211 281 491
31146661 7 13 31 61 181
31405501 71 631 701
31692805 5 47 157 859
32914441 7 19 61 4057
33302401 11 31 61 1601
33596641 13 17 281 541
34196401 17 47 127 337
34657141 191 421 431
34901461 7 19 397 661
35571601 13 31 61 1447
35703361 61 277 2113
36121345 5 13 17 97 337
36765901 37 613 1621
37167361 7 11 41 61 193
37280881 11 17 73 2731
37354465 5 29 73 3529
37964809 109 379 919
38151361 41 53 97 181
38624041 37 61 109 157
38637361 7 37 241 619
39353665 5 13 193 3137
40160737 19 23 29 3169
40280065 5 7 67 89 193
40430401 11 101 151 241
40622401 17 43 61 911
40917241 19 31 127 547
41298985 5 7 13 139 653
41341321 7 19 31 37 271
41471521 7 13 31 61 241
42490801 31 41 101 331
43286881 11 31 61 2081
43331401 43 631 1597
43584481 17 31 191 433
43620409 7 19 157 2089
44238481 7 61 313 331
45318561 3 29 173 3011
45877861 31 43 127 271
45890209 29 53 73 409
46483633 7 19 373 937
47006785 5 7 17 199 397
48321001 37 41 53 601
48628801 13 31 67 1801
49333201 17 61 113 421
50201089 97 673 769
53245921 17 41 79 967
53711113 157 313 1093
54767881 7 37 103 2053
55462177 17 23 83 1709
56052361 211 421 631
58489201 19 29 101 1051
60112885 5 7 443 3877
60957361 61 181 5521
62756641 109 241 2389
64377991 163 487 811
64774081 29 71 163 193
65037817 13 19 73 3607
65241793 29 43 113 463
67371265 5 13 37 109 257
67653433 19 29 199 617
67902031 43 271 5827
67994641 11 13 37 71 181
68154001 151 601 751
69331969 7 19 37 73 193
70561921 31 53 67 641
72108421 7 11 191 4903
72286501 7 67 79 1951
74165065 5 13 59 83 233
75151441 17 19 29 71 113
75681541 13 19 61 5023
75765313 13 29 73 2753
76595761 11 17 31 73 181
77826001 11 43 137 1201
78091201 31 37 103 661
78120001 19 73 151 373
79411201 193 257 1601
79624621 139 691 829

Haskell

Mit dem folgenden Programm für den Haskell-Interpreter *Hugs2001* kann man selbst evaluieren, ob eine gegebene Zahl carmichael ist. Es ist mir bewusst, dass es bessere Methoden für diesen einfachen Test gibt, zugegeben gibt es weitere Eigenschaften für Carmichael-Zahlen, die eine weitaus leichtere Bestimmung dieser zulässt, im Rahmen dieser kurzen Arbeit habe ich allerdings darauf verzichtet.

```
ggt(x,y)   | x == 0   = y
           | y == 0   = x
           | x >  0   = ggt(y,x `mod` y)
carmichael x = [y|y <-[2..(x-1)], ((y^(x-1)) `mod` x == 1)] ==
[y|y <-[2..x], (ggt(x,y)== 1)]
```