

Grosse Primzahlen - wenn der Umweg schneller ist. . .

Armin P. Barth

1 Eine kleine Hoffnung auf 100'000 Dollar

Die RSA Laboratories¹ in den USA veröffentlichen laufend Zahlen, wie sie in aktuellen Verschlüsselungsverfahren benutzt werden, und fordern dazu auf, sie zu faktorisieren. Beispielsweise war am 20. April 2002, als dieser Text entstand, die 309-stellige Zahl

```
135066410865995223349603216278805969938881475605667027524485143851526
510604859533833940287150571909441798207282164471551373680419703964191
743046496589274256239341020864383202110372958725762358509643110564073
501508187510676594629205563685529475213500852879416377328533906109750
544334999811150056977236890927563
```

noch nicht faktorisiert, und es war ein Geldpreis von 100'000 Dollar demjenigen versprochen, dem eine Faktorisierung gelingen sollte. Ob eine Zahl faktorisiert werden kann oder nicht, ist ein für die Kryptographen wichtiger Hinweis darauf, welche Schlüssellängen benutzt werden müssen, um grösstmögliche Sicherheit garantieren zu können.

Freilich gibt es nur ganz wenig Hoffnung, im Rahmen des Mathematikunterrichts am Gymnasium die erwähnten 100'000 Dollar zu gewinnen. Dennoch lässt sich einiges von dem, was die Faszination der modernen Kryptographie ausmacht, in die Schule tragen. Die heutigen Verschlüsselungsverfahren arbeiten mit 200- bis 300-stelligen Primzahlen, und es stellt sich sofort die Frage, wie die Primheit so grosser Zahlen überhaupt festgestellt werden kann. Es leuchtet schnell ein, dass die Frage, ob eine grosse Zahl n prim ist, nicht mehr mit dem klassischen Algorithmus getestet werden kann, der mit "brute force" alle Zahlen $2, 3, 4, \dots, \lfloor \sqrt{n} \rfloor$ nach einem potentiellen Teiler absucht. Ist n nämlich etwa 200-stellig, so wären ungefähr 10^{100} Divisionen auszuführen, was selbst bei einem sehr schnellen Computer mit 10^8 Operationen pro Sekunde unerträglich lange dauern würde, nämlich rund 10^{84} Jahre!

Der in diesem Artikel vorgestellte schnelle Primtest-Algorithmus wurde zusammen mit dem RSA-Verschlüsselungsalgorithmus im Rahmen einer Arbeitswoche mit einer Schulklasse erarbeitet. Natürlich muss erst der dafür nötige Nährboden bereitgestellt werden, d.h. es sollten einige Lektionen für die Erarbeitung des Modulrechnens eingesetzt werden. Dafür kann dann reich geerntet werden: Der unten gezeigte Primtest-Algorithmus ist nur eine mögliche Pflanze, die auf diesem Boden mit geringem Zusatzaufwand aufgezogen werden kann. Daneben besteht die Möglichkeit, auf Verfahren zur Authentifizierung digitaler Unterschriften einzugehen, ein Thema, das allein der Aktualität und Brisanz wegen die Studierenden meist interessiert. Und natürlich kann auch der RSA-Algorithmus, allerdings mit etwas mathematischem Mehraufwand, auf diesem Boden gut wachsen. Für interessierte Kolleginnen und Kollegen sind in Anmerkung 2 Literaturhinweise vermerkt.

Eine riesige Zahl faktorisieren zu müssen, ist bis zum heutigen Tag eine sehr viel schwierigere Aufgabe, als zu testen, ob sie prim ist. Schon deswegen widmen sich die folgenden Abschnitte nun der leichteren Aufgabe. Wenn eine Zahl ihrer Grösse wegen nicht mehr durch "direktes Nachschauen", also durch Divisionen durch ihre Vorgängerzahlen, auf Primheit getestet werden kann, so empfiehlt es sich, nach einem schnelleren Umweg zu suchen. Ein solcher wird in Abschnitt 2, ein weiterer in Abschnitt 4 besprochen.

2 Der „kleine Fermat“ und Zahlen, die wahrscheinlich prim sind

Wenn direktes Nachschauen zu lange dauert, um zu entscheiden, ob n prim ist oder nicht, so besteht ein möglicher Umweg darin, den "kleinen Satz von Fermat" zu benutzen. Dieser besagt folgendes:

Pierre DE FERMAT (1640):

$$n \text{ prim} \Rightarrow a^{n-1} = 1 \pmod{n} \quad \forall a \in \mathbb{Z}_n \setminus \{0\}$$

Ein Beweis dieses Satzes ist in Anmerkung 3 skizziert. Auf den ersten Blick will nicht so recht einleuchten, wie dieser Satz bei einem schnellen Primtest behilflich sein soll, hat er doch die Form einer Implikation, bei der nur dann etwas schlüssig gefolgert werden kann, wenn n schon als prim erkannt ist. Man ist sogar versucht zu denken, dass der Satz nur dann helfen könnte, wenn der Pfeil in die andere Richtung zeigen würde, was er aber nicht tut. (Übrigens wäre gerade dies dem Zweck eines schnellen Primtests nicht dienlich!)

Dies eine ist immerhin klar, dass n nicht prim sein kann, wenn es gelingt, eine einzige Zahl a zu finden, für die $a^{n-1} \neq 1 \pmod{n}$ gilt. Denn wenn n prim wäre, so müssten ja alle Zahlen $a \in \mathbb{Z}_n \setminus \{0\}$ die Eigenschaft $a^{n-1} = 1 \pmod{n}$ besitzen. Und damit ist die wesentliche Richtung des Umweges schon eingeschlagen: Man wählt einige *wenige* Zahlen $a \in \mathbb{Z}_n \setminus \{0\}$ und testet für jede, ob $a^{n-1} = 1 \pmod{n}$ oder nicht. Findet man nur eine einzige Zahl, für die das nicht zutrifft, so kann n unmöglich prim sein. Liefert aber jede Potenz den Wert 1, so kann, genau genommen, nicht gefolgert werden, dass n prim ist; diese Folgerung wäre selbst dann unzulässig, wenn die Potenzen aller Zahlen aus $\mathbb{Z}_n \setminus \{0\}$ bestimmt würden und den Wert 1 lieferten.

Man kann aber vorsichtig vermuten, dass n *wahrscheinlich* prim ist, wenn alle zufällig gewählten Zahlen $a \in \mathbb{Z}_n \setminus \{0\}$ die Eigenschaft $a^{n-1} = 1 \pmod{n}$ besitzen, und dass diese Wahrscheinlichkeit noch zunimmt, wenn mehr Kandidaten a ausgewählt werden. Diese auf SOLOVAY und STRASSEN⁴ zurückgehende Idee ist ebenso halbsbrecherisch wie verführerisch: Wenn n nicht prim ist, sollte sich dies irgendwie zeigen, d.h. es sollte ziemlich rasch ein a gefunden werden, bei dem $a^{n-1} \neq 1 \pmod{n}$ ist. Anders ausgedrückt: Erfüllen einige Testzahlen a die Eigenschaft $a^{n-1} = 1 \pmod{n}$, ohne dass sich eine einzige Ausnahme zeigt, so sollte die Irrtumswahrscheinlichkeit beim Entscheid "n ist prim" sehr klein sein.

Natürlich ist dies noch sehr vage, und natürlich muss über diese Wahrscheinlichkeit weiter unten noch gesprochen werden. Zunächst soll aber der Algorithmus formuliert werden, der über die Primheit einer vorliegenden Zahl n ohne *direktes Nachschauen* auf einem schnelleren Umweg entscheidet:

Algorithmus 1:

Input: $n \in \mathbb{N}$

Schritt 1: Wähle k Zufallszahlen⁵ $a_1, \dots, a_k \in \mathbb{Z}_n \setminus \{0\}$ (z.B. $k = 50$)

Schritt 2: Berechne für alle $i = 1, 2, \dots, k$ den Term $a_i^{n-1} \pmod{n}$.

- Falls das Ergebnis ein einziges Mal $\neq 1$ ist, so verlasse die Schleife und entscheide (mit absoluter Sicherheit): " n ist nicht prim".

- Falls das Ergebnis nie $\neq 1$ ist, entscheide: " n ist wahrscheinlich prim".

3 Fehlentscheide und eine zahme Potenzierung

Algorithmen, deren Entscheide mit einer bestimmten Irrtumswahrscheinlichkeit behaftet sind, heißen *probabilistische Algorithmen*. Der obige Algorithmus hat diesen Nachteil, und es fragt sich, wie stark dieser Nachteil ins Gewicht fällt, ob und wie häufig er falsch entscheidet. Die Vorteile sind aber unübersehbar: Es muss mit lediglich 50 Zahlen ein bestimmter Test

durchgeführt zu werden! Gelingt der Test ein einziges Mal nicht, so ist die vorgelegte Zahl nicht prim mit Irrtumswahrscheinlichkeit 0; gelingt er aber immer, so ist die vorgelegte Zahl wahrscheinlich prim mit einer Irrtumswahrscheinlichkeit, die hoffentlich sehr klein ist.

Zu dem Vorteil, dass lediglich eine Schleife mit etwa 50 Durchgängen ausgeführt zu werden braucht, gesellt sich der weitere Vorteil, dass ein einzelner Test, ob nämlich $a_i^{n-1} = 1 \pmod{n}$ oder nicht, relativ schnell gelingt. Es muss zwar bei jedem Test die $(n-1)$ -te Potenz einer Zahl $\leq n$ berechnet werden, doch da das Ergebnis nur modulo n verlangt wird, ist garantiert, dass die Zahlen nie sehr gross werden. Berechnet man die Potenz nämlich geschickt, so wird kein Zwischenresultat mehr als zweimal so viele Stellen haben wie n . Dies soll an einem einfachen Beispiel erläutert werden:

Angenommen, es soll die allfällige Primheit von $n = 17$ getestet werden und der aktuelle Schleifendurchgang soll den Fermat-Test mit $a = 5$ ausführen. Dann empfiehlt es sich nicht, zuerst die Potenz 5^{16} und danach den Rest bei Division durch 17 zu berechnen, denn die Potenz 5^{16} hat immerhin 12 Stellen. Einfacher ist es, die 16. Potenz als eine vierfache Quadratur aufzufassen und nach jeder einzelnen Quadratur immer sofort den Rest modulo 17 zu bilden. Der Fermat-Test umfasst dann also diese Schritte:

1. $5^2 = 25$; $25 \bmod 17 = 8$
2. $8^2 = 64$; $64 \bmod 17 = 13 (= 5^4 \bmod 17)$
3. $13^2 = 169$; $169 \bmod 17 = 16 (= 5^8 \bmod 17)$
4. $16^2 = 256$; $256 \bmod 17 = 1 (= 5^{16} \bmod 17)$

Und tatsächlich sind alle Zwischenresultate klein geblieben.

Allgemein kann zur Berechnung der Zahl $a^{n-1} \bmod n$ eine auf SCHOLZ⁶ (1937) zurückgehende Idee herangezogen werden, die sicherstellt, dass die Stellenzahl aller Zwischenergebnisse stets kleiner oder gleich der doppelten Stellenzahl von n ist: Dazu wird zuerst der Exponent $n-1$ im Zweiersystem dargestellt:

$$n-1 = b_s \cdot 2^s + b_{s-1} \cdot 2^{s-1} + \dots + b_2 \cdot 2^2 + b_1 \cdot 2$$

wobei alle Koeffizienten entweder gleich 0 oder gleich 1 sind. (Da niemand ernstlich eine gerade Zahl auf Primheit testen dürfte, ist $n-1$ sicherlich gerade, so dass auf den Koeffizienten b_0 verzichtet werden kann!) Danach werden (immer modulo n) der Reihe nach die Zwischenresultate

$$\begin{aligned} z_1 &= a^{b_s \cdot 2} \\ z_2 &= a^{b_s \cdot 2 + b_{s-1}} \\ z_3 &= a^{b_s \cdot 2^2 + b_{s-1} \cdot 2} \\ z_4 &= a^{b_s \cdot 2^2 + b_{s-1} \cdot 2 + b_{s-2}} \\ z_5 &= a^{b_s \cdot 2^3 + b_{s-1} \cdot 2^2 + b_{s-2} \cdot 2} \\ &\vdots \end{aligned}$$

berechnet, bis schliesslich, nach höchstens $2 \cdot \log_2(n)$ Schritten, der Wert von $a^{n-1} \bmod n$ feststeht. Dabei ist jeder einzelne Schritt entweder eine Quadratur einer Zahl $< n$ oder aber eine Multiplikation zweier Zahlen $< n$, so dass in der Tat die Stellenzahl nie grösser sein wird als die doppelte Stellenzahl von n . Wegen $b_0 = 0$ ist die letzte Operation eine Quadratur, was in Abschnitt 4 sehr wichtig sein wird.

Übrigens lässt sich leicht einsehen, dass zur Berechnung einer $(n-1)$ -ten Potenz stets mindestens $\log_2(n-1)$ einzelne Multiplikationen nötig sind, so dass der auf SCHOLZ zurückgehende Algorithmus nicht mehr wesentlich beschleunigt werden kann. In den heute üblichen CAS-Taschenrechnern ist eine schnelle modulare Potenzierung schon implementiert, so dass sie, soll Algorithmus 1 auf einem Taschenrechner erprobt werden, nicht erst programmiert werden muss.

4 Ein noch besserer Umweg dank Lagranges Hilfe

Algorithmus 1 birgt die Gefahr eines Fehlentscheides in sich. Ist n nicht prim und liefern zufällig alle $(n - 1)$ -ten Potenzen der Zufallszahlen a_1, a_2, \dots, a_k den Wert $1 \pmod{n}$, so entscheidet der Algorithmus fälschlicherweise, dass n prim ist. Nun lässt sich mit Hilfe eines Satzes von LAGRANGE ein zusätzlicher Test einführen, welcher in einigen Fällen die Sicherheit von Algorithmus 1 erhöht. Der angekündigte Satz besagt:

Satz von LAGRANGE⁷:

$$n \text{ prim} \Rightarrow a^{\frac{n-1}{2}} \in \{1, n-1\} \pmod{n} \quad \forall a \in \mathbb{Z}_n \setminus \{0\}$$

Ein Beweis ist in Anmerkung 8 skizziert. Wie kann dieser Satz einer Verbesserung von Algorithmus 1 dienen? Nun, da $n-1$ zweifellos gerade ist, sind beide Faktoren in $n-1 = \frac{n-1}{2} \cdot 2$ natürliche Zahlen. Die Berechnung der Potenz a^{n-1} kann also auch so realisiert werden: $a^{n-1} = \left(a^{\frac{n-1}{2}}\right)^2$, wobei $a^{\frac{n-1}{2}}$ das Zwischenresultat unmittelbar vor der letzten Quadratur darstellt.

Und daraus ergibt sich sofort ein weiterer Test: Ist für irgendeine der gewählten Zufallszahlen a_1, a_2, \dots, a_k das Zwischenresultat vor der letzten Quadratur weder die Zahl 1 noch die Zahl $n-1$, so kann n unmöglich prim sein. Wäre n prim, so müsste ja stets das Zwischenresultat vor der letzten Quadratur eine der beiden genannten Zahlen sein. Dieser Zusatztest verbessert die Chance auf einen korrekten Entscheid noch. Ist nämlich n nicht prim und liefern zufällig alle $(n - 1)$ -ten Potenzen der Zufallszahlen den Wert $1 \pmod{n}$, so kann es gut sein, dass wenigstens ein Zwischenresultat vor einer letzten Quadratur weder den Wert 1 noch den Wert $n - 1$ hat; und damit entscheidet der Algorithmus dann richtig: n ist nicht prim.

Schliesst man zusätzlich aus, dass 1 als Zufallszahl gewählt werden kann, was nicht sinnvoll wäre, so erhält man den besseren Algorithmus 2:

Algorithmus 2:
 Input: $n \in \mathbb{N}$
 Schritt 1: Wähle k Zufallszahlen $a_1, \dots, a_k \in \mathbb{Z}_n \setminus \{0\}$ (z.B. $k = 50$)
 Schritt 2: Berechne für alle $i = 1, 2, \dots, k$ den Term $\left(a_i^{\frac{n-1}{2}}\right)^2 \pmod{n}$.
 - Falls das Ergebnis ein einziges Mal $\neq 1$ ist oder falls das Zwischenresultat vor der letzten Quadratur ein einziges Mal weder 1 noch $n - 1$ ist, so verlasse die Schleife und entscheide (mit absoluter Sicherheit): " n ist nicht prim."
 - Falls das Ergebnis nie $\neq 1$ ist und jedes Zwischenresultat vor der letzten Quadratur entweder gleich 1 oder gleich $n - 1$ ist, entscheide: " n ist wahrscheinlich prim".

Damit ist der schnelle Umweg zum Primtest schon sehr robust, und es bleibt die Frage, wie sicher der Entscheid ist, wie sehr man ihm vertrauen kann.

5 Das Austricksen der Spielverderber

Es gibt tatsächlich Zahlen, die Spielverderber sind! Soll etwa getestet werden, ob $n = 9$ prim ist oder nicht, und erwischt man als Zufallszahl ausgerechnet $a = 8$, so würde der erste Algorithmus $a^{n-1} \pmod{n} = 8^8 \pmod{9} = 1$ berechnen und befinden, dass 9 wahrscheinlich prim ist. Allerdings sind $a = 1$ und $a = 8$ die einzigen Spielverderber dieser Art; alle anderen Zahlen aus $\mathbb{Z}_9 \setminus \{0\}$ benehmen sich anständig und liefern bei Potenzbildung eine Zahl $\neq 1$, so dass Algorithmus 1 wahrscheinlich richtig entscheidet.

Die Zahl $a = 8$ ist sogar ein doppelter Spielverderber, weil sie sich auch dem Algorithmus

2 widersetzt: $8^8 \bmod 9 = (8^4)^2 \bmod 9 = 1^2 \bmod 9$; das Zwischenresultat vor der letzten Quadratur ist also gleich 1, so dass auch Algorithmus 2, würde einzig mit der Zufallszahl 8 gearbeitet, zum falschen Schluss kommt.

Soll getestet werden, ob $n = 15$ prim ist, so gibt es vier Spielverderber für Algorithmus 1: $a = 1, a = 4, a = 11$ und $a = 14$. Sie alle liefern beim Fermat-Test den Wert 1 und führen den Algorithmus folglich in die Irre. In Algorithmus 2 allerdings wird $a = 1$ gar nicht mehr zugelassen, und die Spielverderber $a = 4$ und $a = 11$ werden ausgetrickst: $4^{\frac{n-1}{2}} \bmod n = 4^7 \bmod 15 = 4$, und $11^{\frac{n-1}{2}} \bmod n = 11^7 \bmod 15 = 11$. Folglich täuscht nur noch der doppelte Spielverderber $a = 14$ den Algorithmus 2, so dass also, da sicherlich mit mehr als einer Zufallszahl gearbeitet wird, Algorithmus 2 richtig entscheidet.

Schreibt man ein kleines Programm, welches für kleine Zahlen n die doppelten Spielverderber (also Zufallszahlen $a \in \mathbb{Z}_n \setminus \{0, 1\}$, die Algorithmus 2 in die Irre führen) aufspürt, so bemerkt man, dass es erstaunlich wenige davon gibt. SOLOVAY und STRASSEN bewiesen in ihrer Arbeit aus dem Jahr 1977⁹, dass für jede beliebige natürliche Zahl n höchstens die Hälfte aller Zahlen $\leq n$ doppelte Spielverderber sind, so dass also die Wahrscheinlichkeit dafür, dass eine einzelne Zufallszahl a den Algorithmus 2 zu einer falschen Entscheidung verleitet, $\leq \frac{1}{2}$ ist.

Nun wird der Vorschlag, $k = 50$ Zufallszahlen $\leq n - 1$ zu wählen, verständlich. Die Wahrscheinlichkeit, dass der Algorithmus bei 50 Zufallszahlen irrt, ist nämlich $\leq \left(\frac{1}{2}\right)^{50} \approx 9 \cdot 10^{-16}$. Diese Wahrscheinlichkeit ist bedeutend kleiner als die Chance, im Schweizer Zahlenlotto einen Sechser anzukreuzen. Algorithmus 2 ist also sehr sehr sicher. Die Irrtumswahrscheinlichkeit ist schon bei 50 Zufallszahlen vernachlässigbar klein und lässt sich für grösseres k gegen 0 drücken. Es ist ein leichtes, die doppelten Spielverderber mit relativ geringem Aufwand auszutricksen.

Dank Algorithmus 2 kennt die Mathematik also einen Primtest, der die gewünschte Information mit praktisch grenzenloser Sicherheit und ohne direktes Nachschauen auf einem viel schnelleren Umweg beschafft.

6 Anmerkungen

1. RSA Laboratories, 20 Crosby Drive, Bedford, MA 01730-1402 USA, <http://www.rsasecurity.com/rsalabs>
2. J. Buchmann: *Einführung in die Kryptographie*, Springer-Verlag, Berlin (2000)
S. Singh: *Geheime Botschaften*, Carl Hauser-Verlag, München (2000)
Spektrum der Wissenschaft, Dossier: *Kryptographie*
J. Meier: *Einsicht in die Kryptographie*, in: ISTRON-Materialien für einen realitätsbezogenen Unterricht, Band 6(2000), pp. 151–157
A. Beutelspacher: *Geheimssprachen*, C. H. Beck-Verlag, München (1997)
H. Sommer: *Zahlentheorie in der Schule? Der RSA-Algorithmus und seine hochaktuelle Anwendung*, in: PM 4/40. Jg. 1998, pp. 159–162
A.P. Barth: *Algorithmik für Einsteiger*, vieweg-Verlag, Braunschweig (2003)
3. Sei also $a \in \mathbb{Z}_n \setminus \{0\}$ beliebig. Zunächst lässt sich leicht einsehen, dass alle folgenden Produkte modulo n paarweise verschieden sind: $1 \cdot a, 2 \cdot a, 3 \cdot a, \dots, (n-1) \cdot a$. Deswegen und weil alle diese Zahlen ≥ 1 und $\leq n-1$ sind, bleibt nur der Schluss übrig, dass es sich dabei gerade um die Zahlen $1, 2, 3, \dots, n-1$ handelt, freilich nicht unbedingt in dieser Reihenfolge. Deshalb ist

$$[1 \cdot 2 \cdot 3 \cdot \dots \cdot (n-1)] \bmod n = [(1 \cdot a) \cdot (2 \cdot a) \cdot (3 \cdot a) \cdot \dots \cdot ((n-1) \cdot a)] \bmod n$$
und folglich:

$$((n-1)!) \bmod n = [a^{n-1} \cdot (n-1)!] \bmod n$$
also

$$[(a^{n-1} - 1) \cdot (n-1)!] \bmod n = 0.$$
Und da n unmöglich Teiler von $(n-1)!$ sein kann, muss n Teiler von $a^{n-1} - 1$ sein.
4. R. Solovay, V. Strassen: *A fast Monte-Carlo test for primality*, Siam J. Comput. Vol.6, No.1, March 1977, pp. 84–85

5. Bei der Auswahl von Zufallszahlen muss darauf geachtet werden, dass keine Zahl eine Potenz einer anderen ist; erfüllt nämlich irgendeine Zahl den Fermat-Test, so auch jede Potenz dieser Zahl.
6. A. Scholz: *Aufgabe 253*, Jber. d. Dt. Math. Verein. 92, class II 47, 1937, pp. 41–42
7. Allgemeiner besagt dieser Satz, dass bei primem n jede Zahl $\leq n - 1$ höchstens 2 Quadratwurzeln hat. Da dies auch für die Zahl 1 gilt und da 1 und $n - 1$ natürlich Quadratwurzeln von 1 sind, folgt der im Text behauptete Satz leicht.
8. Sei also n prim und sei $a \leq n - 1$. Da n ungerade ist, ist $\frac{n-1}{2}$ Element von \mathbb{Z}_n . Nun gilt $\left(a^{\frac{n-1}{2}} + 1\right) \cdot \left(a^{\frac{n-1}{2}} - 1\right) = a^{n-1} - 1 = 0 \pmod{n}$ nach kleinem Fermat. Da das Produkt zweier Zahlen im Körper \mathbb{Z}_n nur dann 0 sein kann, wenn mindestens einer der beiden Faktoren gleich 0 ist, folgt, dass entweder $a^{\frac{n-1}{2}} + 1 = 0$ oder $a^{\frac{n-1}{2}} - 1 = 0$ sein muss. Im zweiten Fall ist $a^{\frac{n-1}{2}} = 1 \pmod{n}$ und im ersten Fall ist $a^{\frac{n-1}{2}} = -1 = n - 1 \pmod{n}$.
9. a.a.O.