Seminar „Sicherheit im Internet“

Faktorisierung großer Zahlen

von Jan Kirchhoff
(Email: jkirchhoff@gmx.de)

Betreuer: Prof. Dr. H. Kuchen

Inhaltsverzeichnis

1. Einleitung

Die Faktorisierung beschäftigt man sich mit der Zerlegung von Zahlen in ihre Primfaktoren.

Die Sicherheit heute üblicher Public/Private-Key-Verschlüsselungsverfahren wie das häufig genutzte RSA-Verfahren und des Rabin-Verfahren basiert auf dem Problem, dass bis heute kein Algorithmus bekannt ist, mit dem es möglich wäre, Zahlen, die das Produkt sehr großer Primfaktoren sind, in akzeptabler Zeit zu faktorisieren.

Um das Problem zu verdeutlichen: Es ist aus heutiger Sicht eher unproblematisch, nachzuweisen, dass eine Zahl, die aus 1000 Dezimalstellen besteht, eine Primzahl ist, aber es ist heutzutage praktisch unmöglich, eine Zahl mit 200 Dezimalstellen, die das Produkt zweier Primzahlen ist, zu faktorisieren.

Im RSA-Verfahren werden Zahlen erzeugt, deren Teiler aus zwei etwa gleich großen Primzahlen bestehen. Die größte Zahl, die mit diesem Verfahren erzeugt wurde und faktorisiert werden konnte, hat 130 Dezimalstellen oder 432 Bit Schlüssellänge.

Im folgendem sollen einige der Algorithmen vorgestellt werden, die heutzutage für die Faktorisierung von Zahlen genutzt werden. Wir gehen dabei bei allen Verfahren davon aus, dass die zu faktorisierende Zahl keine Primzahl ist.

2. Die Probedivision

Die offensichtlichste Möglichkeit, die es gibt, um eine Zahl n zu faktorisieren, besteht darin, diese Zahl probeweise durch alle Primzahlen p zu dividieren, für die gilt, dass p kleiner als n/2 ist. Dabei beginnt man üblicherweise mit der kleinsten Primzahl, der zwei und dividiert die Zahl n so oft durch diese Zahl, bis das Ergebnis teilerfremd wird. Anschließend wählt man die nächste Primzahl und setzt den Vorgang solange fort, bis die Zahl vollständig zerlegt werden konnte.

Dabei ergeben sich zwei Probleme:

- Bei Faktorisierung einer sehr großen Zahl müssen möglicherweise sehr viele Primzahlen ermittelt werden oder gespeichert vorliegen.

- Des weiteren muss die Zahl n probeweise durch alle diese Primzahlen geteilt werden, bis sie in ihre Faktoren zerlegt werden kann.

Nach einer Abschätzung von Rosser und Schoenfeld gibt es unter einer Zahl $x \geq 18$ etwa $\pi(x) = \frac{x}{\log x}$ Primzahlen. Sollte versucht werden, eine Zahl mit 150 Stellen, die aus zwei etwa gleichgroßen Primzahlen besteht, (wie z.B. die Zahlen, die beim RSA-Verfahren auftreten), durch die Probedivision zu faktorisieren, müssten für eine vollständige Analyse mit diesem Verfahren etwa $\frac{10^{75}}{\log 10^{75}} = 5{,}79 \cdot 10^{72}$ Primzahlen getestet werden. Es dürfte damit offensichtlich sein, dass ein solcher Test praktisch nicht durchführbar ist, und die Probedivision für „Angriffe“ auf moderne Verschlüsselungsverfahren unbrauchbar ist.

Allerdings wird dieses Verfahren häufig genutzt, um bei einer zu faktorisierenden sehr großen Zahl kleine Primfaktoren zu „eliminieren“, es werden also üblicherweise alle Primteiler unter einer Schranke wie z.B. 10^6 rausdividiert. Anschließend wird die Zahl mit Hilfe eines anderen Verfahrens vollständig zerlegt.

Beispiel:

Faktorisierung der Zahl 23595.

1. Schritt: Testen auf die erste Primzahl, die 2 ➔ 23595 ≡1 mod 2 ➔ 2 ist nicht Primteiler.
2. Schritt: Testen auf die zweite Primzahl, die 3 ➔ 23595 ≡0 mod 3 ➔ 3 ist Primteiler, wir teilen 23595 durch 3 und rechnen mit dem Ergebnis 7865 weiter.

3. Schritt: Erneutes Testen auf die Primzahl 3, da sie mehrfacher Teiler der Zahl sein kann ➔ 7865 ≡2 mod 3 ➔ 3 ist nur ein mal Primteiler.

4. Schritt: Testen auf 5 ➔ 7865 ≡0 mod 5 ➔ 5 ist Primteiler, wir rechnen mit 7865 : 5 = 1573 weiter.

5. Schritt: Erneut testen auf 5 ➔ 1573 ≡3 mod 5 ➔ 5 ist nur ein mal Primteiler.

6. Schritt: Testen auf 7 ➔ 1573 ≡5 mod 7 ➔ 7 ist nicht Primteiler.

7. Schritt: Testen auf 11 ➔ 1573 ≡0 mod 11 ➔ 11 ist Primteiler, wir rechnen mit 1573 : 11 = 143 weiter.

8. Schritt: Erneut testen auf 11 ➔ 143 ≡0 mod 11 ➔ 11 ist zwei mal Primteiler, wir rechnen mit 143 : 11 = 13 weiter.

9. Schritt: Testen auf 13 ➔ 13 = 0 mod 13 ➔ 13 ist Primteiler der Zahl, da 13 : 13=1 ist, ist die Zahl faktorisiert.

Die Zahl 23595 ist also aus 3 · 5 · 7 · 11 · 11 · 13 zusammengesetzt.

3. Die (p-1)-Methode

Die Zahl direkt „unter“ einer beliebigen Primzahl *p* größer *5*, also der Wert *(p-1)*, ist offensichtlich keine Primzahl, sondern ist aus einzelnen Faktoren zusammengesetzt, in jedem Fall aus der Zahl *2*, mit 50%tiger Wahrscheinlichkeit aus der Zahl *3* usw.

Bei der *(p-1)*-Faktorisierungsmethode hofft man, dass diese „Vorgängerzahl“ einer der Primteiler der zu faktorisierenden Zahl n aus möglichst kleinen Primteilern zusammengesetzt ist, da es dann möglich ist, die Zahl mit Hilfe des kleinen Satz von Fermat relativ schnell zu faktorisieren.

Der kleine Satz von Fermat besagt, dass der Ausdruck $a^{p-1} - 1$ bei beliebigem $a \in N$ für jede beliebige Primzahl $p \in$ P durch diese restlos teilbar ist, sofern *a* nicht selbst durch *p* teilbar ist. (P ist die Menge aller Primzahlen.)

Wenn wir nun z.B. durch Raten alle Primfaktoren von *p-1* ermittelt haben, so ist $a^{p-1} - 1$ durch die uns ggf. unbekannte Primzahl *p* teilbar. Sollte *p* nun ein Teiler von *n* sein, so ist der größte gemeinsame Teiler von $a^{p-1} - 1$ und der zu faktorisierenden Zahl *n* gleich dieser Primzahl – wir können also in diesem Fall mit relativ geringem Aufwand einen Teiler von *n* finden.

Da der Wert von *a* praktischerweise beliebig gewählt werden kann, kann der Wert *k* von a^k auch ein Vielfaches von dem gesuchten Wert *p–1* sein, da bei $k = v \cdot (p-1)$ bei $a^k = a^{v \cdot (p-1)} = (a^v)^{p-1}$ natürlich auch für die „Basis“ a^v gilt, dass diese beliebig gewählt werden kann. Dies ist recht praktisch, da im Exponenten Primfaktoren vorhanden sein können, die nicht Bestandteil des gesuchten Primteilers „minus eins“ sind.

Dies führt dazu, dass bei einem Test auf den größten gemeinsamen Teiler von $a^{p-1}-1$ und *n* der Wert von *n* implizit auf sehr viele Primzahlen *p* getestet wird - nämlich auf alle, die sich durch eine beliebige Kombination der Primfaktoren des Exponenten *p-1* von a^{p-1} ergeben – da $a^{p-1}-1$ durch alle diese Primfaktoren teilbar ist. (Ausgenommen sind dabei natürlich alle Primzahlen *p*, die bei $(a^v)^{p-1}$ auch durch a^v teilbar sind.)

Es ist allerdings nicht sichergestellt, das wir bei diesem Verfahren in jedem Fall einen Primteiler erhalten, es kann ggf. erforderlich sein, die resultierende Zahl erneut zu faktorisieren, da $a^{p-1}-1$ mehrere Teiler der zu faktorisierenden Zahl *n* enthalten kann.

Da wir später ’nur’ den größten gemeinsamen Teiler von $a^{p-1}-1$ und der zu untersuchenden Zahl *n* berechnen, kann in der Moduloklasse *n* gerechnet werden. Dies kann damit begründet werden, dass es bei Berechnung des größten gemeinsamen Teilers erlaubt ist, den kleineren Wert vom größeren Wert abzuziehen – schließlich ist die Differenz zwischen diesen beiden Zahlen ebenfalls ein Vielfaches des größten gemeinsamen Teilers, hat jedoch in keinem Fall einen größeren gemeinsamen Teiler mit einer der beiden Werte.

Wenn wir nun also in der Moduloklasse *n* rechnen, entspricht das dem fortwährendem Abziehen des kleineren Teilers n vom größeren Teiler $a^{p-1}-1$, bis dieser kleiner *n* wird. Also ist ggt($a^{p-1}-1$,*n*) = ggt($a^{p-1}-1$ mod *n*,*n*).

Dies ist sehr praktisch, da für Potenzoperationen für die Berechnung von a^{p-1} innerhalb von Moduloklassen stark vereinfachende Rechenregeln wie z.B. die Methode des fortschreitenden Quadrierens existieren. Mit dieser ist es sehr schnell und unproblematisch möglich, sehr komplexe Ausdrücke wie z.B. $2^{232.792.560}$ innerhalb einer Moduloklasse zu berechnen – die in dieser Größenordnung bei der Multiplikation vieler Primfaktoren im Exponenten sehr schnell auftreten.

Für die Berechnung des größten gemeinsamen Teilers kann z.B. durch den euklidischen oder den binären GGT-Algorithmus erfolgen, siehe dazu [Rei1].

Für a wird üblicherweise eine kleine Zahl gewählt, z.B. die 2. Es gibt verschiedene Ansätze, den Exponenten von a zu bestimmen: Eine Möglichkeit ist es, *a* einfach mit einem geratenen Exponenten *k* fortwährend zu potenzieren und anschließend jeweils einen Test mit ggt(*a-1* mod *n*,*n*) auf einen Teiler ungleich eins auszuführen. Sollte sich ein Wert ungleich eins ergeben, so haben wir einen Teiler gefunden.

Eine etwas trickreichere Variante stammt von Pollard und basiert darauf, die Anzahl der recht aufwendigen ggt-Tests zu reduzieren, in dem bei der Anwendung des Algorithmus in einem Schritt eine größere Anzahl von ausgewählten Primfaktoren zugefügt wird.

Dafür wird für alle Primfaktoren *p* unter einer Schranke *B* ein Exponent *v* gesucht, für den gilt, dass $p^v \leq B$ und $p^{v+1} > B$ ist. Anschließend werden die Produkte der potenzierten Primzahlen zu k_B aufmultipliziert:

$$k_B = \prod_{p \in P, p^v \leq B, p^{v+1} > B} p^v$$

Beispiel:

Wir wählen die Schranke B = 10. Da $2^3 = 8$ und $2^4 = 16$ ist, folgt daraus, dass $v_2=3$ ist. Für die Primzahl 3 gilt, dass $3^2=9$ und $3^3=27$ ist. Daher wird hier $v_3=2$ gewählt. Für die Primzahlen 5 und 7 wird v_5 bzw. v_7 mit 1 angesetzt. k_{10} ist also gleich $2^3 \cdot 3^2 \cdot 5 \cdot 7 = 2520$.

Darauf hin wird die Zahl $a^{k_B} \bmod n$ berechnet und mit *n* auf einen gemeinsamen Teiler ungleich *1* getestet. Sollte dieser Test nicht erfolgreich enden, so wird das Verfahren mit einer anderen (in der Regel höheren) Schranke oder einem anderen *a* erneut ausgeführt.

Beispiel:

Wir möchten die Zahl n = 864371 faktorisieren.

Als Schranke wählen wir B=20. k ist in diesem Fall gleich $2^4 \cdot 3^2 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19$ = 232.792.560. Wir berechnen (mit Hilfe der Methode des quadratischen Fortschreibens) den Wert von a^k mod n, dies ergibt 794803.

Anschließend berechnen wir ggt(a^k -1 mod n,n) = ggt(794802, 864371) = 953. Wir haben einen Primteiler von n gefunden.

Dies konnte uns aber nur gelingen, da der Vorgänger von 953, die Zahl 952 = 2·2·2·7·17 nur aus Primfaktoren besteht, die kleiner als B, also 20 sind.

Zur Faktorisierung der Zahl n = 733763 müsste die Schranke wesentlich höher gewählt werden, da die Zahl n das Produkt der Primzahlen 809·907 ist. Die Primfaktoren der Zahl 808 sind 2·2·2·101, die Faktoren der Zahl 906 sind 2·3·151. In diesem Fall müsste die Schranke B also mindestens auf 101 gesetzt werden, was die Berechnung erheblich aufwendiger machen würde.

An dem vorherigen Beispiel erkennt man, dass es bei der *(p-1)*-Methode keine Erfolgsgarantie gibt. Je höher man die *B*-Schranke wählt, desto größer ist die Erfolgsaussicht, aber der für die Berechnung zu treibende Aufwand steigt ebenfalls enorm an.

Für „Angriffe“ auf heutige auf dem Faktorisierungsproblem basierende Algorithmen wie z.B: RSA ist diese Methode ungeeignet, da in heutigen Anwendungen die Vorgängerzahl der verwendeten Primzahlen überprüft werden, ob sie mit diesem Verfahren angreifbar sind, d.h. vollständig aus relativ kleinen Primfaktoren zusammengesetzt sind. Sollte dies der Fall sein, so wird diese Primzahl verworfen.

4. Das quadratische Sieb

Das quadratische Sieb stellt einen der am weitesten entwickelten Algorithmen zur Faktorisierung dar.

Er basiert darauf, zwei Zahlen x und y für die zu faktorisierende Zahl n zu finden, für die gilt, dass

$x^2 \equiv y^2 \bmod n$

und

$x \not\equiv y \bmod n$ bzw. $x \not\equiv -y \bmod n$ ist.

In diesem Fall gilt also, dass x^2-y^2 ein Vielfaches von n ist. Gleichzeitig ist n weder ein Teiler von (x-y) noch von (x+y). Nach dem dritten binomischen Gesetz ist $x^2-y^2 = (x-y)\cdot(x+y)$.

Wenn x^2-y^2 durch n teilbar ist, dann müssen (x-y) und (x+y) aus Vielfachen der Teiler von n zusammengesetzt sein, damit $x^2-y^2 = (x-y)\cdot(x+y)$ und $x^2 \equiv y^2 \bmod n$ gelten kann.

Also ist, wenn beide obere Bedingungen gelten, der größte gemeinsame Teiler von n und x-y, also der ggt(n,x-y), ein Teiler von n. (Dies gilt natürlich auch für ggt(n,x+y), da aber x+y erheblich größer als x-y ist und in (x+y) nur die in (x-y) fehlenden Primteiler von n enthalten sind, wird üblicherweise darauf verzichtet, auch diesen größten gemeinsamen Teiler zu ermitteln – eine Division von n durch den ggt(n,x-y) liefert das gleiche Ergebnis in wesentlich kürzerer Rechenzeit.)

Beispiel:

Wir möchten die Zahl n = 10441 faktorisieren.

Sei x = 2022 und y = 840, dann gilt, dass $x^2 \equiv y^2 \bmod 10441$ ist. Gleichzeitig gilt offensichtlich, dass $x \not\equiv \pm y \bmod 10441$ ist.

Der größte gemeinsame Teiler von x-y und n, also ggt(1182,10441) ist gleich 197. Dies ist ein Primteiler von n, wobei sich bei diesem Verfahren nicht notwendigerweise Primteiler ergeben müssen, falls n aus mehreren Primteilern zusammengesetzt wurde. Es kann also notwendig sein, das resultierende Ergebnis weiter zu faktorisieren.

Für die Bestimmung der Variablen x und y bestimmen wir zuerst den Wert *m* mit $m = \lfloor \sqrt{n} \rfloor$.

Dann sei $f(s) = (s+m)^2 - n$.

Nun werden Zahlen *s* gesucht, für die gilt, dass der Wert von *f(s) mod n* aus möglichst kleinen Primfaktoren (und eventuell *-1*, falls die resultierende Zahl negativ ist) besteht. Die Höhe der Primfaktoren wird dabei durch eine Zahl *B* begrenzt, dabei werden alle Werte von *f(s)* mit einem oder mehreren Primfaktoren größer *B* ignoriert. Die Zahlen, die nur Primfaktoren kleiner oder gleich *B* haben, nennt man *B*-glatt.

Beispiel:

Für die von uns zu faktorisierende Zahl n = 10441 ergibt sich $m = \lfloor \sqrt{n} \rfloor$ *mit 102.*

Die sich daraus ergebenden Werte für f(s) sind:

s	*s+m*	*f(s)=(s+m)²-n*	*f(s) faktorisiert*
...	*...*	*...*	*...*
-5	*97*	*-1032*	$-1 \cdot 2^3 \cdot 3 \cdot 43$
-4	*98*	*-837*	$-1 \cdot 3^3 \cdot 31$
-3	***99***	***-640***	$\mathbf{-1 \cdot 2^7 \cdot 5}$
-2	***100***	***-441***	$\mathbf{-1 \cdot 3^2 \cdot 7^2}$
-1	***101***	***-240***	$\mathbf{-1 \cdot 2^4 \cdot 3 \cdot 5}$
0	*102*	*-37*	$-1 \cdot 37$
1	***103***	***168***	$\mathbf{2^3 \cdot 3 \cdot 7}$
2	***104***	***375***	$\mathbf{3 \cdot 5^3}$
3	*105*	*584*	$2^3 \cdot 73$
4	*106*	*795*	$3 \cdot 5 \cdot 53$
5	***107***	***1008***	$\mathbf{2^4 \cdot 3^2 \cdot 7}$
...	*...*	*...*	*...*
19	***121***	***4200***	$\mathbf{2^3 \cdot 3 \cdot 5^2 \cdot 7}$
...	*...*	*...*	*...*

Bei einem B = 10 sind alle hier fett markierten Werte B-glatt. (Demzufolge haben alle nicht markierten Werte zumindest einen Primfaktor größer 10.)

Sollten die *B*-glatten *f(s)*-Werte ermittelt worden sein, so werden Kombinationen der Zahlen *f(s)* gesucht, für die die Summe der Exponenten jeder der oben ermittelten Primfaktoren $p_{s,j}$ und der Zahl *-1* gerade oder gleich null ist.

Dies wird genutzt, um eine Kombination von Faktoren zu erhalten, aus denen man unproblematisch die Wurzel ziehen kann, und trotzdem eine natürliche Zahl erhält – wenn die resultierende Zahl positiv ist und die Vielfachen der Primfaktoren p^x einen geraden Exponenten *x* haben, kann man einfach alle Exponenten der Primfaktoren durch zwei teilen und erhält so die Wurzel der Zahl.

Beispiel:

$$\sqrt{2^6 \cdot 3^2 \cdot 5^2 \cdot 7^2} = \sqrt{705600} = 840 = 2^3 \cdot 3 \cdot 5 \cdot 7$$

Fortsetzung des Beispieles von oben:

Nach kurzer Suche in oberer Tabelle, die bei diesem einfachen Beispiel noch manuell erfolgen kann, ergibt sich eine Kombination der f(s)-Gleichungen mit s=1 und s=19, bei der alle Primfaktoren mit geraden Exponenten vorkommen: (Es können natürlich auch mehr als zwei f(s)-Werte Bestandteil einer Kombination sein.)

Nun gilt mit $(s+m)^2 \equiv f(s) \bmod n$*:*

$(103 \cdot 121)^2 \equiv 2^3 \cdot 3 \cdot 7 \cdot 2^3 \cdot 3 \cdot 5^2 \cdot 7 \bmod n$

$(103 \cdot 121)^2 \equiv 2^6 \cdot 3^2 \cdot 5^2 \cdot 7^2 \bmod n$

$(103 \cdot 121)^2 \equiv (2^3 \cdot 3 \cdot 5 \cdot 7)^2 \bmod n$

Gleichzeitig gilt:

$103 \cdot 121 = 2022 \not\equiv 840 = 2^3 \cdot 3 \cdot 5 \cdot 7 \bmod n$

Damit haben wir Werte für x mit 2022 und für y mit 840 gefunden, da für diese Werte gilt, dass $x^2 \equiv y^2 \bmod n$ *sowie* $x^2 \not\equiv \pm y^2 \bmod n$ *ist. Diese haben wir bereits oben verwendet.*

B-glatte Werte werden verwendet, um die Wahrscheinlichkeit zu steigern, schnell eine gültige Kombination zu finden, für die alle Exponenten der Primteiler gerade oder null sind, da das mehrfache Auftreten von einzelnen sehr großen Primzahlen sehr unwahrscheinlich ist.

Um automatisiert eine Kombination von Gleichungen zu ermitteln, wird jeder *B*-glatten Gleichung eine Variablen λ_s zugeordnet, die den Wert eins annehmen, wenn die zugehörige Gleichung *f(s)* der Kombination zugehörig ist bzw. null annimmt, wenn sie nicht zugehörig ist.

Daraus ergibt sich für jede der in Betracht gezogene Primzahl $p_i \leq B$ (und der *-1*) eine eigene Gleichung der Form:

$d_{-1,s1} * \lambda_{s1} + d_{-1,s2} * \lambda_{s2} + d_{-1,s3} * \lambda_{s3} + \ldots \equiv 0 \bmod 2$ für -1,

$d_{2,s1} * \lambda_{s1} + d_{2,s2} * \lambda_{s2} + d_{2,s3} * \lambda_{s3} + \ldots \equiv 0 \bmod 2$ für die Primzahl $p_1 = 2$,

$d_{3,s1} * \lambda_{s1} + d_{3,s2} * \lambda_{s2} + d_{3,s3} * \lambda_{s3} + \ldots \equiv 0 \bmod 2$ für die Primzahl $p_2 = 3$,

$d_{5,s1} * \lambda_{s1} + d_{5,s2} * \lambda_{s2} + d_{5,s3} * \lambda_{s3} + \ldots \equiv 0 \bmod 2$ für die Primzahl $p_3 = 5$ usw.

wobei s_x die Indizes der faktorisierten B-glatten Wertpaare sind, $d_{-1,h}$ den Exponenten von *-1* und $d_{p,s}$ den Exponenten des Primteilers $p \in P$ von $f(s) = (s+m)^2 - n$ angibt. (*P* ist die Menge aller Primzahlen.)

Beispiel:

(Zur Vereinfachung beachten wir nur die Werte von s mit -1,1,2 und 19, das Beispiel würde sonst zu umfangreich werden.)

λ_{-1} *gehört zu f(-1)=* $\mathbf{-240 = -1 \cdot 2^4 \cdot 3 \cdot 5,}$ λ_1 *gehört zu f(1)=* $\mathbf{168 = 2^3 \cdot 3 \cdot 7,}$
λ_2 *gehört zu f(2)=* $\mathbf{375 = 3 \cdot 5^3,}$ λ_{19} *gehört zu f(19)=* $\mathbf{4200 = 2^3 \cdot 3 \cdot 5^2 \cdot 7}$

Dadurch ergeben sich folgende Gleichungen:

Primzahl/Vorfaktor	*Gleichung*
-1	$1 \cdot \lambda_{-1} + 0 \cdot \lambda_1 + 0 \cdot \lambda_2 + 0 \cdot \lambda_{19} \equiv 0 \bmod 2$
2	$4 \cdot \lambda_{-1} + 3 \cdot \lambda_1 + 0 \cdot \lambda_2 + 3 \cdot \lambda_{19} \equiv 0 \bmod 2$
3	$1 \cdot \lambda_{-1} + 1 \cdot \lambda_1 + 1 \cdot \lambda_2 + 1 \cdot \lambda_{19} \equiv 0 \bmod 2$
5	$1 \cdot \lambda_{-1} + 0 \cdot \lambda_1 + 3 \cdot \lambda_2 + 2 \cdot \lambda_{19} \equiv 0 \bmod 2$
7	$1 \cdot \lambda_1 + 0 \cdot \lambda_1 + 0 \cdot \lambda_2 + 1 \cdot \lambda_{19} \equiv 0 \bmod 2$

oder gekürzt:

Primzahl/Vorfaktor	*Gleichung*
-1	$\lambda_{-1} \equiv 0 \bmod 2$
2	$4 \cdot \lambda_{-1} + 3 \cdot \lambda_1 + 3 \cdot \lambda_{19} \equiv 0 \bmod 2$
3	$\lambda_{-1} + \lambda_1 + \lambda_2 + \lambda_{19} \equiv 0 \bmod 2$
5	$\lambda_{-1} + 3 \cdot \lambda_2 + 2 \cdot \lambda_{19} \equiv 0 \bmod 2$
7	$\lambda_1 + \lambda_{19} \equiv 0 \bmod 2$

Die Koeffizienten $d_{n,sm}$ können modulo zwei geteilt werden, da alle quadratischen Exponenten für unsere Betrachtung irrelevant sind – wir müssen nur Primzahlen beachten, für die uns noch das „Gegenstück“ fehlt.

Beispiel:

Die Tabelle verändert sich entsprechend:

Primzahl/Vorfaktor	*Gleichung*
-1	$\lambda_{-1} \equiv 0 \bmod 2$
2	$\lambda_1 + \lambda_{19} \equiv 0 \bmod 2$
3	$\lambda_{-1} + \lambda_1 + \lambda_2 + \lambda_{19} \equiv 0 \bmod 2$
5	$\lambda_{-1} + \lambda_2 \equiv 0 \bmod 2$
7	$\lambda_1 + \lambda_{19} \equiv 0 \bmod 2$

Wenn man dieses Gleichungssystem auflöst, ergibt sich z.B.:

$\lambda_{-1} = \lambda_2 = 0$ *und* $\lambda_1 = \lambda_{19} = 1$

(Es ergibt sich auch die Kombination, in der alle Werte $\lambda_x = 0$ *sind. Diese ist natürlich für uns uninteressant.)*

Damit wissen wir, dass die 1. und 19. Gleichungen zu wählen ist..

Bei diesem einfachen Gleichungssystem ist es unproblematisch möglich, eine Lösung manuel zu finden. Sollten jedoch sehr große Zahlen faktorisiert werden, muss man möglicherweise mit vielen Millionen Gleichungen umgehen, um eine Lösung zu finden.

Das Auflösen dieses Gleichungssystem kann z.B. durch dass Gauß-Verfahren erfolgen, in der Regel werden jedoch spezialisierte Verfahren für dünn besetzte Gleichungssysteme wie z.B. das Verfahren von Wiedemann genutzt.

Anzumerken ist noch, wie die Überprüfung auf B-Glätte der f(s)-Werte erfolgt: Dazu wird die Zahl nicht einfach durch alle in Betracht kommenden Primfaktoren dividiert, sondern in dem sie durch ein spezielles Verfahren sehr trickreich und effizient „gesiebt“ wird:

Dazu wird zuerst ein Siebintervall festgelegt, in dem die Werte für s überprüft werden sollen. Die Breite von diesem wird mit C festgelegt, dabei gilt, dass $-C \leq s \leq C$ ist.

Nun wird ausgenutzt, dass das Polynom $f(s)=[(s+m)^2-n] \bmod p$ in einem Intervall der Länge p maximal zwei Nullstellen besitzen kann – also maximal zwei Werte durch p teilbar sind. Die gefunden Nullstellen gelten mit ihrer Position auch relativ für ihre Nachbarintervalle – man kann also in Schritten der Länge p nach „links“ und „rechts“ gehen und dabei sicher sein, ebenfalls einen Wert zu haben der zumindest einmal durch p teilbar ist.

Beweis:

Wenn gilt, dass $(m + s)^2 - n \equiv 0 \bmod p$ *und* $m,s,n \in N, p \in P$,
dann ist auch $(m + s + k \cdot p)^2 - n \equiv 0 \bmod p$ *und* $k \in N$, *da*

$(m + s + k \cdot p)^2 - n \equiv 0 \equiv$
$((m + s) + k \cdot p)^2 - n \equiv 0 \equiv$
$(m+s)^2 + 2 \cdot (m+s) \cdot k \cdot p + k \cdot p^2 - n \equiv 0 \bmod p$

Da $2 \cdot (m+s) \cdot k \cdot p$ *und* $k \cdot p^2$ *jeweils ganze Vielfache von p sind, gilt:*

$(m+s)^2 + 2 \cdot (m+s) \cdot k \cdot p + k \cdot p^2 - n \equiv (m + s)^2 - n \bmod p$

Damit ist $(m + s + k \cdot p)^2 - n \equiv (m + s)^2 - n \equiv 0 \bmod p$.

Sollte also in dem Intervall der Länge p in $f(s) \bmod p$ eine Nullstelle gefunden werden, so wird der Wert solange durch die Primzahl p dividiert, bis der resultierenden Wert teilerfremd zu p ist. Das gleiche erfolgt im gesamten Siebintervall mit allen Werten, deren Position jeweils genau die Intervallänge $n \cdot p$, $n \in \mathrm{N}$ entfernt ist.
Dabei kann bei der ersten Division aller Werte, die um die Schrittlänge $n \cdot p$ entfernt liegen, davon ausgegangen werden, dass der Wert auf jeden Fall einmal durch p teilbar ist.
Anschließend wird das Intervall nach einer zweiten Nullstelle durchsucht – sollte sie gefunden werden, wird mit dieser genau so verfahren wie mit der ersten Nullstelle. Nach Ermittlung der zweiten Nullstelle kann das Verfahren für die betrachtete Primzahl beendet werden.

Beispiel:

Wir berechnen das Sieb für unser oberes Beispiel mit n = 10441.

s	*(s-m)²-n*	*Sieb mit 2*	*Sieb mit 3*	*Sieb mit 5*	*Sieb mit 7*
-8	*-1605*		*-535*	*-107*	
-7	*-1416*	*-177*	*-59*		
-6	*-1225*			*-49*	***-1***
-5	*-1032*	*-129*	*-43*		
-4	*-837*		*-31*		
-3	*-640*	*-5*		***-1***	
-2	*-441*		*-49*		***-1***
-1	*-240*	*-15*	*-5*	***-1***	
0	*-37*				
1	*168*	*21*	*7*		***1***
2	*375*		*125*	***1***	
3	*584*	*73*			
4	*795*		*265*	*53*	
5	*1008*	*63*	*7*		***1***
6	*1223*				
7	*1440*	*45*	*5*	***1***	
8	*1659*		*553*		*79*

Man erkennt sehr leicht an den markierten Blöcken, die jeweils die Länge der Primzahl besitzen, dass offensichtlich in jedem Block die teilbare Zahl an der gleichen Position aufzufinden ist.

B-glatt für B = 10 sind hier alle Werte, die in einer Spalte den Wert 1 oder -1 stehen haben, also für s die Werte -6,-3,-2,-1,1,-2,-5 und -7. Diese Werte konnten mit den Primfaktoren 2,3,5 und 7 vollständig zerlegt werden.

Der Vorteil an diesem Sieb-Verfahren ist nun, dass man sich eine große Menge unnötiger Probedivisionen sparen kann und man maximal ein Intervall der Länge der Primzahl p durchsuchen muss.

Hier gibt es wieder stärker optimierte Verfahren für dieses Sieb, da die Suche durch Ausprobieren bei sehr großen Primzahlen sehr aufwendig wird.

5. Analyse des quadratischen Siebs

Die Bestimmung der Laufzeit des quadratischen Siebes ist nicht ganz einfach.

Offensichtlich ist, dass der Aufwand mit der größe der zu faktorisierenden Zahl n tendenziell steigt und damit von diesen abhängt. Man geht bei dieser Laufzeitanalyse davon aus, dass der Aufwand von der Anzahl der Dezimalstellen (oder der Anzahl der zum Speichern der Zahl benötigten Bits) abhängt. Daher wird die Laufzeit für den Logarithmus der zu faktorisierenden Zahl n bestimmt.

Häufig kann die Laufzeit eines Algorithmus in gewisse Kategorien eingeordnet werden, z.B. lineare Laufzeit, polynominelle Laufzeit oder exponentielle Laufzeit. Die Probedivision ist z.B. ein Beispiel für exponentielle Laufzeit.

Die polynominelle Laufzeit wird üblicherweise mit $L_n[v] = n^v$ dargestellt, wobei der so genannte Komplexitätskoeffizient v erst später bestimmt wird.

Die exponentielle Laufzeit wird durch $L_n[v] = e^{v \cdot n}$ dargestellt, wobei v hier wieder der Komplexitätskoeffizient ist.

Da wir die Laufzeit anhand der Anzahl der Dezimal-/Binärstellen festlegen möchten, geben wir die polynominelle Laufzeit als $L_n[v] = (\log n)^v$ und die exponentielle Laufzeit als $L_n[v] = e^{v \log n}$ an.

Die Laufzeit des quadratischen Siebes ist zwischen der exponentiellen und der polynominellen Laufzeit anzusiedeln. Um dies abzubilden, wird folgende Gleichung verwendet:

$$L_n[u,v] = \left(e^{v \cdot \log n}\right)^u \cdot \left[(\log n)^v\right]^{1-u} =$$
$$e^{v \cdot \log n \cdot u} \cdot (\log n)^{v \cdot (1-u)} =$$
$$e^{v \cdot (\log n)^u \cdot (\log \log n)^{1-u}}$$

Hier gibt der Koeffizient u an, welchem Bereich sich die Laufzeit annähert: Bei $u = 0$ geht man von polynominelle Laufzeit aus, im Bereich zwischen $0 < u < 1$ spricht man subexponentieller Laufzeit, bei $u = 1$ handelt es sich um exponentielle Laufzeit

Es ist bisher nicht gelungen, die Laufzeit des quadratischen Siebes endgültig zu bestimmen, da Annahmen über die Verteilung der *B*-glatten Zahlen innerhalb der *f(s)*-Werte, die zur Bestimmung der Laufzeit notwendig sind, bisher nicht bewiesen werden konnten. (Die Annahmen scheinen aber von empirischen Untersuchungen bestätigt zu werden.)

Man vermutet nun, dass die Anzahl der *B*-glatten Werte innerhalb *f(s)* in der gleichen Größenordnung wie die Anzahl der *B*-glatten Werte aller natürlicher Zahlen kleiner $\sqrt{n}$ ist.

Die Anzahl der B-glatten natürlichen Zahlen, die kleiner als x sind, bezeichnet man mit $\psi(x,B)$.
Es konnte bewiesen werden, dass wenn $x \geq 10$ und $w \leq (\log x)^{1-e}$ ist, dann ist die abgeschätzte Anzahl der $x^{1/w}$-glatten Zahlen gleich $xw^{-w+f(x,w)}$.

$$\psi(x, x^{1/w}) = x^{w-w+f(x,w)}$$

f(x,w) ist dabei eine Funktion, für die gilt, dass $f(x,w)/w \rightarrow 0$ konvergiert, wenn $w \rightarrow \infty$ strebt. (Annäherungsweise kann man davon ausgehen, dass die Anzahl der $x^{1/w}$-glatten Zahlen in etwa gleich $x \cdot w^{-w}$ ist.)

Es kann weiterhin davon ausgegangen werden, dass $f(s)=(s+m)^2-n$ mit $m = \lfloor \sqrt{n} \rfloor$ bei kleinem *s* im Verhältnis zu *n* recht klein ist, da m^2-n im Verhältnis zu n ebenfalls recht klein ist.

Nun kann gezeigt werden, dass für positive reele Zahlen *a*, *u* und *v* gezeigt werden kann,
dass $\psi(n^a, L_n[u,v]) = n^a \cdot L_n\left[1-u, -\frac{a}{v}(1-u)+o(1)\right]$ ist.

Beweis:

$$L_n[u,v] = e^{v\cdot(\log n)^u \cdot(\log\log n)^{1-u}} = n^{v\left(\frac{(\log\log n)}{\log n}\right)^{1-u}}$$

Setzt man $w = \frac{a}{v}\left(\frac{\log n}{\log\log n}\right)^{1-u}$ *und wendet das oben angegebene Theorem über* $\psi(x, x^{1/w})$ *an,*

dann erhält man: $\psi(n^a, L_n[u,v]) = n^a w^{-w(1+o(1))}$

Dabei ist $w^{-w(1+o(1))} = \left(e^{(1-u)\left(\log\frac{a}{v}+\log\log n-\log\log\log n\right)\left(-\frac{a}{v}\left(\frac{\log n}{\log\log n}\right)^{1-u}(1+o(1))\right)} \right)^{1-u}$.

Hierin ist $\log\frac{a}{v} + \log\log n - \log\log\log n = \log\log n(1+o(1))$.

Daher ist $w^{-w(1+o(1))} = e^{(\log n)^{1-u}(\log\log n)^{u}\left(-\frac{a}{v}(1-u)+o(1)\right)} = L_n\left[1-u,-\frac{a}{v}(1-u)+o(1)\right]$

Der Koeffizient *a* gibt dabei den Exponenten der Größenordnung der von dem Algorithmus genutzten Zahlen an, diese sind beim quadratischen Sieb gleich $\sqrt{n}$. Also wird *a* mit $\frac{1}{2}$ festgelegt.

Um ein s zu finden, für das f(s) $L_n[u,v]$-glatt ist, braucht man demzufolge in etwa $L_n\left[1-u,\frac{1}{2v}(1-u)-o(1)\right]$ Elemente im Siebintervall. Die Anzahl der Elemente der Faktorbasis, also alle Primzahlen kleiner $B = L_n[u,v]$ ist höchstens $L_n[u,v]$, also braucht man eben so viele Elemente *s*, damit das Gleichungssystem lösbar wird, um hinterher ebenso viele Gleichungen wie Unbekannte zu erhalten. Je größer *u* ist, desto größer wird das Gleichungssystem, gleichzeitig sinkt die Größe des Siebintervalls mit *1-u*. Um die Zeit für das Sieben und das Gleichungslösen auszugleichen, wählt man $u = 1/2$.

Damit benötigen wir für jeden $L_n[u,v]$-glatten s-Wert $L_n\left[\frac{1}{2},\frac{1}{4v}\right]$ Elemente. Da wir insgesamt zur Lösung des Gleichungssystems $L_n\left[\frac{1}{2},v\right]$ Gleichungen benötigen, ist die Größe des Siebintervalls mit $L_n\left[\frac{1}{2},\frac{1}{4v}\right]\cdot L_n\left[\frac{1}{2},v\right] = L_n\left[\frac{1}{2},v+\frac{1}{4v}\right]$ anzusetzen.

Folgende mathematisch-rechnerisch aufwendigen (und damit relevanten) Schritte sind für die Anwendung des quadratischen Siebes auszuführen:

- Die Berechnung der Quadratwurzeln von *f(x)* mod *p* ist in der Faktorbasis ist in Polynomzeit möglich. Die Berechnung aller Wurzeln sollte also in $Ln[1/2, v+o(1)$ Zeiteinheiten möglich sein.
- Die Siebzeit pro Primzahl *p* ist in der Ordnung $O\left(\frac{L_n\left[\frac{1}{2}, v+\frac{1}{4v}+o(1)\right]}{p}\right)$, da man in Schritten der Länge *p* durch das Intervall $L_n[1/2,v]$ gehen muss. Die gesamte Siebzeit inklusive der Vorberechnungen hat dann in etwa die Laufzeit $L_n\left[\frac{1}{2},v+\frac{1}{4v}+o(1)\right]$.
- Das Auflösen des Gleichungssystemes mit dem Algorithmus von Wiedmann für dünn besetzte Gleichungssysteme benötigt etwa $L_n\left[\frac{1}{2},2v+o(1)\right]$ Zeiteinheiten. Durch die

Festsetzung des Wert $v = \frac{1}{2}$ wird die Siebzeit minimiert und wiederum mit der Zeit für das Lösen des Gleichungssystems gleichgesetzt.

Wir erhalten insgesamt eine Laufzeitabschätzung mit $L_n\left[\frac{1}{2}, 1+o(1)\right]$.

6. Effizienz anderer Faktorisierungsverfahren

In diesem Kapitel soll die Frage behandelt werden, ob es vergleichbar oder noch effizientere Faktorisierungsalgorithmen gibt.

- Der effizienteste Faktorisierungsalgorithmus, dessen Laufzeit bewiesen werden konnte, ist ein probabilistischer Algorithmus mit erwarteter Laufzeit $L_n[1/2,1+o(1)]$, also mit der gleichen Laufzeit wie das quadratische Sieb.

- Die Laufzeit der Elliptische-Kurven-Methode hängt nicht von n, sondern von dem kleinsten Primfaktor p der Zahl n ab. Diese ist $L_p[1/2, \sqrt{1/2}]$. Sollten die Primfaktoren einer Zahl n relativ klein sein, so hat diese Methode eine deutlich kürzere Laufzeit als das quadratische Sieb, sollten beide Teiler jedoch in der Größenordnung von $\sqrt{n}$ liegen, kann man von einer Laufzeit der Elliptischen-Kurven-Methode mit $L_n\left[1/2, 1+o(1)\right]$ ausgehen, was also in etwa der der Laufzeit des quadratischen Siebes entspricht. Allerdings ist letzteres in der Praxis in diesem Fall in der Regel schneller. Da bei auf dem Faktorisierungsproblem basierenden Verschlüsselungsverfahren die beiden Primzahlen in vergleichbaren Größenordnungen angesiedelt sind, bringt die Elliptische-Kurven-Methode hier keinen Vorteil.

- 1988 wurde von Pollard das Zahlenkörpersieb entdeckt. Dies basiert auf der algebraischen Zahlentheorie. Dabei wurde eine Möglichkeit entdeckt, die schon aus dem quadratischen Sieb bekannten Variablen x und y systematisch aus relativ kleinen Zahlen zusammenzusetzen. Der Aufwand, der dabei zu treiben ist, ist wesentlich geringer als beim quadratischen Sieb – die (bisher unbewiesene Laufzeit) wird unter geeigneten Annahmen mit $L_n[1/3,(64/9)^{1/3}]$ angenommen und kommt dabei mit u = 1/3 wesentlich näher an einem Polynomzeitalgorithmus heran wie andere bekannten Verfahren.

Im April 1994 konnte eine aus dem RSA-Verfahren entnommene 428-Bit-Zahl mit einem Rechenaufwand von 5000 MIPS-Jahren faktorisiert werden. Dazu wurde das quadratische Sieb benutzt. (1 MIPS-Jahr bedeutet, dass ein Rechner, der eine Millionen Instruktionen pro Sekunde berechnen kann, ein Jahr laufen muß.)

Zwei Jahre später, im April 1996 wurde mit Hilfe des Zahlenkörpersiebes eine größere 432-Bit-Zahl in 500 MIPS-Jahren faktorisiert. Es war also mit diesem neueren Verfahren möglich, eine größere Zahl in einem Zehntel der ursprünglich benötigten Zeit zu faktorisieren.

Es gab in der letzten Jahren weitere große Fortschritte in dem Bereich der Faktorisierung. Bisher ist noch kein Algorithmus entdeckt worden, mit dem es möglich wäre, die Faktoren einer Zahl in polynomineller Laufzeit zu bestimmen – oder er ist nicht bekannt geworden.

Sollte er existieren und entdeckt worden sein, ist es nicht unwahrscheinlich, dass er nicht bekannt werden würde – schließlich würde es dem Entdecker doch „erhebliche Vorteile“ bringen, die Entdeckung geheim zu halten, da z.B. die Sicherheit von Verfahren wie SSL und PGP auf RSA und damit auf dem Faktorisierungsproblem basieren. Diese Verfahren wären in dem Fall als unsicher anzusehen.

7. Literaturverzeichnis

[Buch] Johannes Buchmann, Einführung in die Kryptographie, 2001, Springer, Kap. 8

[Wint] Arne Winterhof, Vorlesungsskript Zahlentheoretische Methoden in der Kryptografie, Mai 2002, Im Internet verfügbar unter www.dismat.oeaw.ac.at/wint/ws01.ps

Internet:

[Rei1] Thorsten Reinecke, Faktorisierung natürlicher Zahlen, Juni 1999, http://www.heydernet.de/information/faktorisierung/faktorisierung.html

[Rei2] Thorsten Reinecke, Gedanken zur (p-1)-Faktorisierungsmethode, Juni 1999, http://www.heydernet.de/information/faktorisierung/p-1.html

[Dorn] Michael Dorner, Der kleine fermatsche Satz, Januar 2002, http://www.zum.de/Faecher/Materialien/dorner/manuskripthtml/kleinerfermat/kleinerfermat.html