Zeit-Experimente zur Faktorisierung

Ein Beitrag zur Didaktik der Kryptologie

von Ralph-Hardo Schulz und Helmut Witten

Ausgangslage

Bei der Übertragung von vertraulichen Nachrichten mit dem RSA-Kryptosystem (vgl. z.B. Witten/Schulz, 2006–2010, oder Schulz, [2]2003) werden diese durch geeignetes Potenzieren und Rechnung modulo n verschlüsselt; hierbei ist n Teil des öffentlichen Schlüssels und als Produkt von zwei geheim zu haltenden Primzahlen p und q gewählt. Die Sicherheit der Verschlüsselung steht und fällt dabei mit der Un-Möglichkeit, n in seine beiden Primfaktoren zu zerlegen. Während das Multiplizieren zweier Zahlen (mit Langzahlarithmetik) sehr schnell geht, ist der Zeitaufwand bzw. überhaupt die Möglichkeit der Faktorisierung von der Ziffernlänge von p und q und damit von n abhängig. Es sollte also $n = p \cdot q$ so gewählt werden, dass n nicht in praktikabler Zeit faktorisierbar ist und damit nicht der Einwegfunktions-Charakter der Verschlüsselung verloren geht. Das Bundesamt für Informationstechnik (BSI) schreibt dazu (BSI, 2009):

> Bei asymmetrischen Verfahren sollte die Mechanismenstärke so gewählt werden, dass die Lösung der zu Grunde liegenden mathematischen Probleme einen unvertretbar großen bzw. praktisch unmöglichen Rechenaufwand erfordert (die zu wählende Mechanismenstärke hängt daher vom gegenwärtigen Stand der Algorithmik und der Rechentechnik ab).

Da der von Experten für die Faktorisierung von 1024-Bit-RSA-Moduli vorhergesagte Aufwand von circa 2^{80} Operationen mit Fortschreiten der Rechentechnik allmählich in den Bereich des technisch Machbaren geriete, so das BSI, sollten für langfristige Sicherheitsanwendungen 2048-Bit-RSA-Moduli eingesetzt werden. In der Praxis werden (z.B. zur Sicherung von WLAN-Netzen) zurzeit sogar schon Module der Länge 4096 Bit verwendet.

Als grober Gradmesser für die Grenzen der zurzeit möglichen Zerlegbarkeit können die bei der Lösung der Aufgaben der sogenannten *RSA-Challenge* erzielten Ergebnisse dienen (vgl. RSA Laboratories, 2010): Mit enormen Anstrengungen konnte ein Team von 13 Wissenschaftlern die Zahl RSA-768 (eine Zahl mit 768 binären bzw. 232 dezimalen Stellen) faktorisieren; nach ihren Angaben hätte das Faktorisieren auf einem herkömmlichen PC rund 2000 Jahre gedauert (vgl. Kleinjung u.a., 2010).

Auch RSA-640 (mit 193 Dezimalstellen) ist faktorisiert, hingegen nicht RSA-704 (mit 212 Dezimalstellen) und RSA-896 (mit 270 Dezimalstellen). In dem in LOG IN erschienenen Artikel *Primfaktorzerlegung – Experimente zum Zeitaufwand* (Schulz, 1996) wurde versucht, das Phänomen des enormen Anstiegs der Rechenzeiten der Faktorisierung bei zunehmender Ziffernlänge von n mit der uns damals zur Verfügung stehenden Rechenkapazität nachzuempfinden. Im vorliegenden Beitrag beschreiben wir eine Aktualisierung mithilfe der Software-Systeme *SAGE* (vgl. SAGE, 2010; vgl. auch die Kurzanleitung in Esslinger u.a., 2010, S.247ff.) und *CrypTool* (vgl. CrypTool, 2010; Version 1.4.30, im Folgenden mit CT1 bezeichnet), bzw. der neuen Version CrypTool 2.0 (hier mit CT2 bezeichnet, siehe auch CT2 in der Literaturliste).

Bei *SAGE* werden viele hochqualifizierte mathematische Open-Source-Pakete kombiniert, die man z.B. online nutzen kann. *CrypTool* ist ebenfalls eine freie Software, die die Konzepte der Kryptografie und der Kryptoanalyse erfahrbar macht. *CrypTool* ist weltweit das verbreitetste Lernprogramm dieser Art (vgl. auch Esslinger/Koy, 2009).

Ziel

Das Ziel unserer Experimente ist, bei der Faktorisierung von $n = p \cdot q$ den Anstieg der Rechenzeiten in Abhängigkeit von der Ziffernlänge ℓ von n mit allgemein zugänglichen (und weitgehend kostenfreien) Mitteln (online auf dem *SAGE*-Notebook oder auf eigenem PC mit *CrypTool*) experimentell zu untersuchen. (Dabei erreicht man natürlich nicht die eben erwähnten Spitzenleistungen wie bei RSA-768.)

Für die dezimale Ziffernzahl $\ell(n)$ von $n = p \cdot q$ gilt dabei

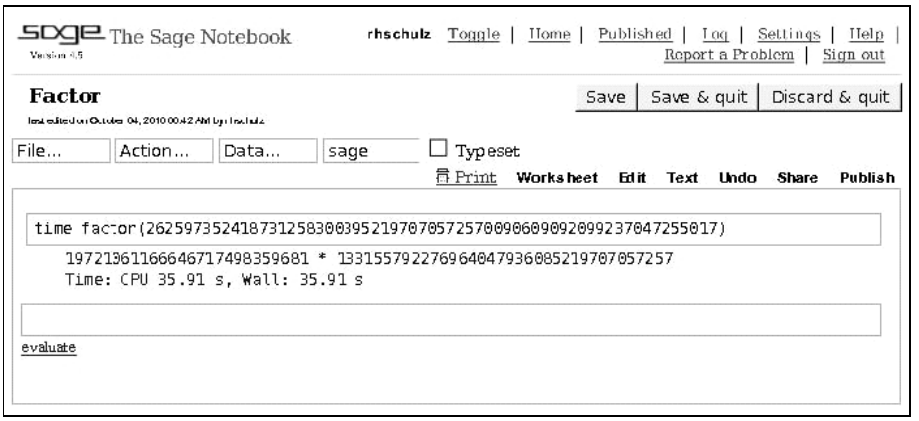

Bild 1: Maske beim Faktorisieren bei *SAGE*.

$\ell(n) = \ell(p)+ \ell(q)-1$ oder $\ell(n) = \ell(p)+ \ell(q)$.

Denn mit $p = p_1 \cdot 10^{\ell(p)-1}$ und $q = q_1 \cdot 10^{\ell(q)-1}$ ist $1 \le p_1,q_1 < 10$ und damit $p \cdot q = p_1 \cdot q_1 \cdot 10^{[\ell(p)+l(q)-1]-1}$ von der Länge $\ell(p)+\ell(q)-1$, falls $1 < p_1 \cdot q_1 < 10$ oder $\ell(p)+ \ell(q)$, falls $10 \le p_1 \cdot q_1 < 100$.

Vorgehen

Zum Erreichen unseres Ziels haben wir unterschiedlich große „Semiprimzahlen“ n eingegeben, also Zahlen, die jeweils Produkt zweier Primzahlen sind. Bei *SAGE* lautet der Befehl time factor(n) (unter *SAGE* selbst, nicht unter PYTHON; siehe Bild 1). Wenn man nicht speziellere Faktorisierungsverfahren (wie das Quadratische Sieb oder ECM, die Elliptische Kurven-Methode) verlangt, so wird das Paket von *PARI* aufgerufen, das ebenfalls Sieb- und ECM-Algorithmen implementiert (laut Beschreibung des Befehls factor bei *SAGE*).

Für die Verwendung von *CrypTool 1* (CT1) sollte man das Programmpaket mit Windows oder einem Windows-Emulator herunterladen und dann die Menüs „Einzelverfahren → RSA-Kryptosystem → Faktorisieren einer Zahl“ benutzen. Bei Problemen kann jederzeit die Hilfe durch die Taste F1 aufgerufen werden.

In CT1 sind folgende Faktorisierungsalgorithmen implementiert (siehe Bild 2): Brute-Force-Methode, Algorithmus nach Brent, Pollards (p–1)-Methode, Williams (p+1)-Methode, Algorithmus nach Lenstra (ECM) und Quadratische Sieb-Methode (QS). Beschreibungen dieser Methoden findet man z. B. in Wikipedia (siehe Literaturliste).

Man beachte auch hier die Hilfe von CT1, bei der einige Kommentare zu den Methoden stehen: So wird empfohlen, Brute Force und Brent als Vorstufe zu verwenden, da diese Verfahren sehr schnell kleine Faktoren finden. Pollards (p–1)- bzw. Williams (p+1)-Methode führe mit großer Wahrscheinlichkeit zum Erfolg, wenn die zu faktorisierende Zahl einen Primfaktor p derart hat, dass $(p–1)$ bzw. $(p+1)$ nur aus kleinen Primfaktoren besteht.

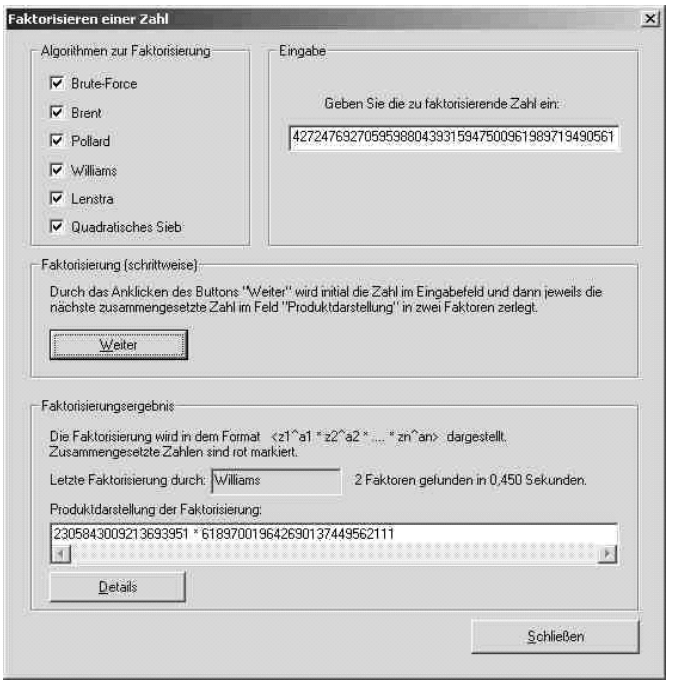

Bild 2: Maske zum Faktorisieren bei *CrypTool 1*.

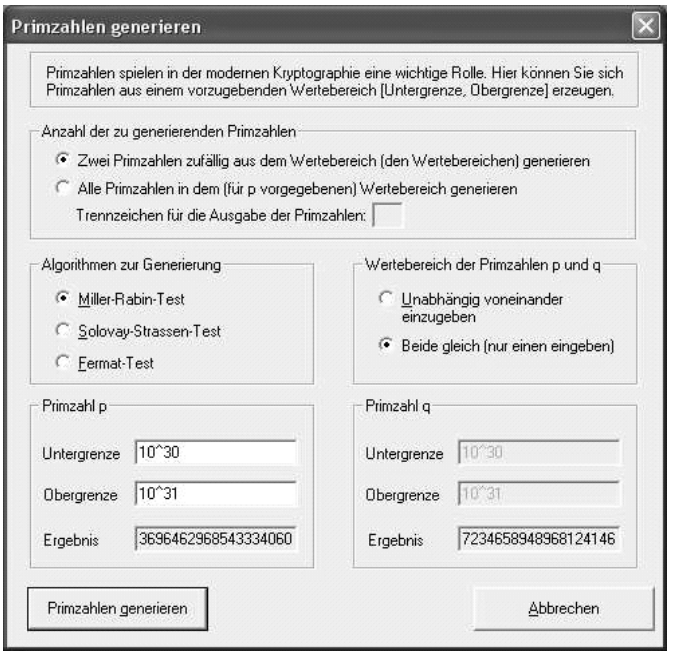

Bild 3: Maske zum Generieren von Primzahlen bei *CrypTool 1*.

Bild 4: Bildschirmfoto nach erfolgter Zerlegung von RSA-100 durch *CrypTool 2*.

Und ECM sei für Faktoren mit bis zu 30 Dezimalstellen geeignet, hingegen das Quadratische Sieb zur Faktorisierung von Zahlen, die aus mehr als einem großen Faktor (mit mehr als 30 Stellen) zusammengesetzt sind.

Alle diese Algorithmen kann man zunächst gleichzeitig anwenden; sie werden dann nach und nach abgeschaltet, wenn sie nicht zum Erfolg führen. Man kann sie aber auch einzeln starten, was wir dann im zweiten Durchgang mit den jeweils erfolgreichen Algorithmen ausgeführt haben. Ein Problem vor dem Starten der Faktorisierung ist die Bereitstellung von geeigneten Faktoren für Semiprimzahlen, d.h. die *Gewinnung von Primzahlen geeigneter Ziffern-Anzahl*. Beim RSA-System sollten im Übrigen die Primzahlen p und q weder zu weit auseinander noch zu nahe beieinander sein; denn wenn einer der beiden Faktoren wesentlich kleiner ist, so ist die Wahrscheinlichkeit der Faktorisierung bei gegebener Länge $\ell(n)$ größer, ebenso falls beide Faktoren nahe beieinander liegen: Aus $p = a–x$ und $q = a+x$ folgt $b(x) := p \cdot q + x^2 = a^2$, sodass für „laufende" x nur zu prüfen ist, ob $b(x)$ Quadrat ist („Faktorisierungsmethode von Fermat").

Die Primzahlen für unsere Experimente lassen sich experimentell z.B. durch Faktorierung von zufällig gewählten Zahlen oder Verwendung von Mersenne-Zahlen – d.h. Zahlen der Form $2^p–1$ mit p prim (vgl. z.B. Ribenboim, ^{3}1996) – gewinnen.

Gezielter kann man auch mit dem Befehl p=next_prime(m);p (bei *SAGE*) oder nextprime(m); (z.B. in *MAPLE*) arbeiten, mit dem sich die kleinste Primzahl oberhalb m bestimmen lässt; eine Zufallsvariante stellt p=next_prime(randint($10^{\ell-1}$, 10^{ℓ}));p dar. Bei *CrypTool* ist die Gewinnung der Primzahlen vorgegebener Größe noch einfacher, nämlich mit den Menüpunkten „Einzelverfahren → RSA-Kryptosystem → Primzahlen generieren" (siehe Bild 3, vorige Seite). Die Semiprimzahlen erhält man dann in sehr kurzer Rechenzeit durch Produktbildung: $n = p \cdot q$.

Bei *CrypTool 2* (CT2) wird zum Faktorisieren eine modifizierte *msieve*-Bibliothek verwendet (zu *msieve* siehe Literatur und Internetquellen am Ende dieses Beitrags). Die Unterstützung von beliebig vielen Kernen in CT2 wurde von Sven Rech zur *msieve*-Bibliothek hinzugefügt. Unter CT2 verwendet *msieve* hauptsächlich das sogenannte MPQS (***M**ultiple **P**olynomial **Q**uadratic **S**ieve*, vgl. z.B. Wikipedia, Stichwort „Quadratisches Sieb"). Dieses Sieb wird allerdings erst angewandt, nachdem mit einigen „einfacheren" Algorithmen versucht wurde, die Zahl möglichst schnell zu zerlegen. (Dieser Versuch dauert ungefähr 1 bis 2 Sekunden.) Erst wenn das nicht gelingen sollte, wird das MPQS verwendet. Zu den einfacheren Algorithmen zählen hier triviale Divisionen (durch kleine Primzahlen), Pollards ρ-Methode (vgl. z.B. Wikipedia, Stichwort „Pollard-Rho-Methode"), P+–1 und die Elliptische Kurven-Methode ECM.

Bei CT2 ist (durch das eingebundene P2P-Framework) auch verteiltes paralleles Rechnen möglich; damit lässt sich (gleichartige Rechner vorausgesetzt) die zum Faktorisieren benötigte Rechenzeit durch die Anzahl der beteiligten Rechner teilen.

Ergebnis

Das Ergebnis unserer Experimente ist in den untenstehenden Tabellen (S. 111 f.; Tabelle 1 für *SAGE*-, Tabellen 2 und 3 für *CrypTool*-Berechnungen) und ausführlich in der durch den LOG-IN-Service zu erhaltenden Tabelle 4 angegeben und grafisch – mithilfe von *Excel* – dargestellt, und zwar in den Bildern 5, 7 und 9, S. 110–111 (Zerlegungszeit t in Abhängigkeit von der dezimalen Ziffernzahl $z = \mathrm{l}(n)$ von n) sowie in den Bildern 6, 8 und 10, S. 110–111 (mit $\ln(t)$ als Funktion von z, also halb-logarithmisch). Man sieht bei den Bildern 6 und 10 deutlich und beim Bild 8 mit Ausreißern, dass der Anstieg der Rechenzeit bei der Faktorisierung ungefähr exponentiell erfolgt und damit sehr schnell an die Kapazitätsgrenze des verwendeten Rechners stößt.

Diskussion

Bei der Rechnung auf *SAGE*, die stets mit *PARI* ausgeführt wurde, zeigt Bild 6 (nächste Seite) mit halblogarithmischen Koordinaten nur geringe Abweichungen von dem (durch eine Gerade angezeigten) exponentiellen Verlauf. Es stört weder, dass die Abweichungen der Längen der Faktoren zum Teil nicht ganz klein sind noch dass gelegentlich Mersenne-Primzahlen (also Primzahlen $M_p = 2^p–1$ mit p prim) genommen wurden. Unklar ist, ob die geringfügig unterschiedlichen Re-

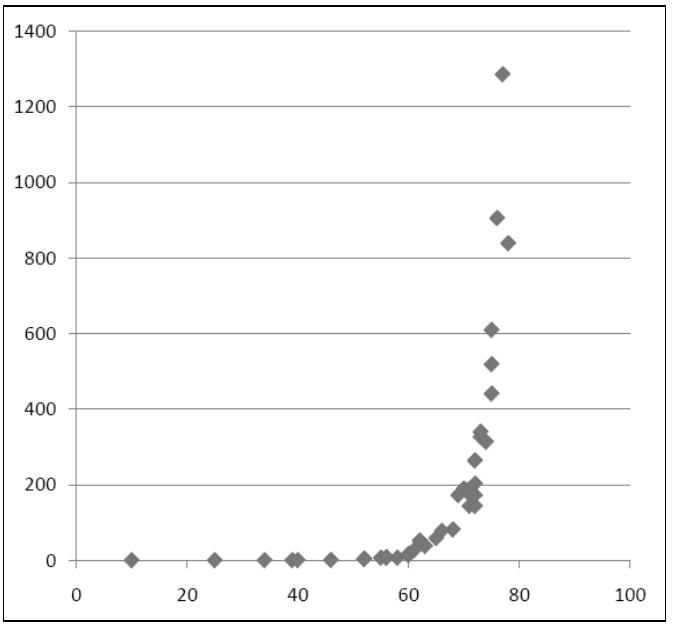

Bild 5: Faktorisierungszeit *t* in Abhängigkeit von der Zeichenlänge von *n* (Rechnung mit *SAGE*).

chenzeiten für ein und dieselbe Zerlegungsaufgabe bei gleichem Algorithmus (siehe die Anmerkungen in Tabelle 1, nächste Seite, bzw. in der über den LOG-IN-Service zu erhaltenden Tabelle 4, siehe S. 143) dadurch entstehen, dass der verwendete Algorithmus nicht ganz deterministisch arbeitet, oder dass dies zu den normalen Zeitabweichungen je nach Auslastung bei Multiuser/Multitasking-Betriebssystemen gehört.

Bei der Rechnung mit CT1 werden die größeren Ausreißer verständlich, wenn man die Verfahren mit berücksichtigt; sie sind nämlich alle bei Pollards (p–1)-Algorithmus bzw. Williams (p+1)-Verfahren aufgetreten, die – wie schon erwähnt – günstig sind, wenn die zu faktorisierende Zahl einen Primfaktor p hat, sodass (p–1) bzw. (p+1)

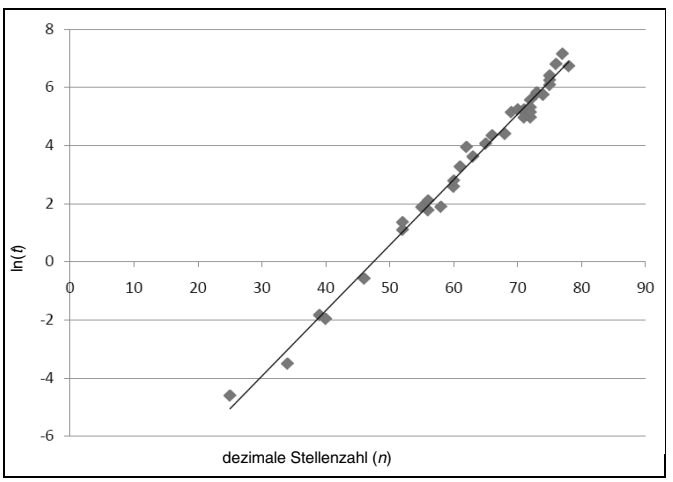

Bild 6: ln(*t*) in Abhängigkeit von der Zeichenlänge von *n* (Rechnung mit *SAGE*).

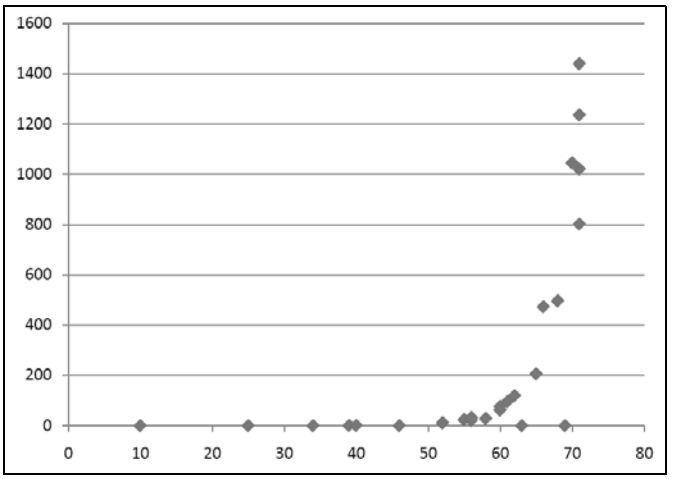

Bild 7: Fatorisierungszeit *t* in Abhängigkeit von der Zeichenlange von *n* (Rechnung mit *CrypTool 1* auf eigenem PC).

nur Produkt kleiner Primfaktoren ist. So hat z.B. bei unserer Zahl der Länge 63 die Zahl p–1 nur Faktoren der Länge 1, 3, 4, 5, 6 und 13. Außerdem sind bei allen großen Abweichungen Mersenne-Primzahlen involviert (bei denen ja $M_p + 1 = 2^p$ die erwähnte Eigenschaft hat). Alle stärkeren Ausreißer waren Ausreißer nach unten, d.h. führten zu einer schnelleren Faktorisierung. Dies zeigt: Die Fälle, in denen Spezialalgorithmen eine Chance haben, sind die Ausnahme; andererseits verlangen diese Sonderfälle, dass man bei der Wahl von Primzahlen Tests durchführt, bevor man sie für RSA benutzt.

Erwartungsgemäß verringern sich die Rechenzeiten bei der Rechnung mit CT1, wenn man (im 2. Durchgang) das günstigste Verfahren fest auswählt.

Eine wesentlich größere Leistungsfähigkeit und daraus resultierende Reduktionen der Rechenzeiten zeigen

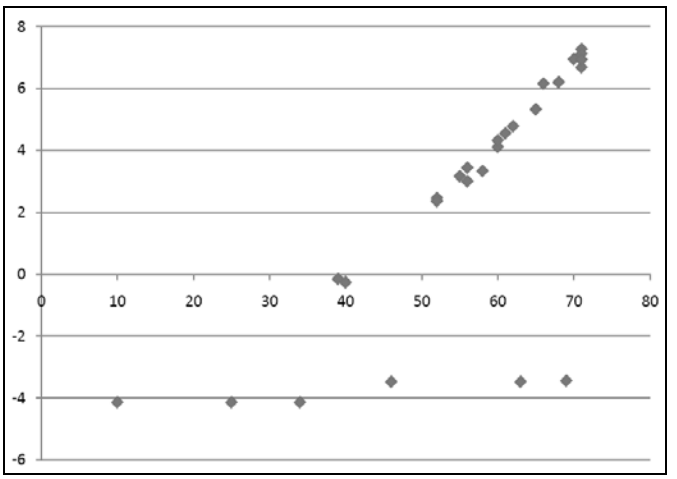

Bild 8: ln(*t*) in Abhängigkeit von der Zeichenlänge von *n* (Rechnung mit *CrypTool 1* auf eigenem PC); stärkere Ausreißer nach unten bei den Methoden von Pollard und Williams.

l(*n*)	*t*(*n*)	ln(*t*(*n*))	*l*(*p*)	*l*(*q*)	Anmerkungen
10	0		5	5	(Jevons-Zahl)
25	0,01	–4,605170	13	13	auch 0,02 s
34	0,03	–3,506558	15	19	
39	0,16	–1,832581	20	20	
40	0,14	–1,966113	20	20	
46	0,56	–0,579818	19 M	27 M	
52	3,00	1,098612	26	26	
52	3,91	1,363537	26	26	
55	6,59	1,885553	28	28	
56	5,88	1,771557	28	28	
56	8,24	2,109000	28	29	
58	6,64	1,893112	27	31	auch 6,63 s
60	13,24	2,583243	27 M	33 M	auch 13,21 und 13,42 s
60	16,33	2,793004	27	33	auch16,28 und 16,38 s
61	26,56	3,279406	26	36	auch 35,91 s
62	52,09	3,952973	31	31	
63	37,42	3,622205	31	33 M	auch 37,37 s
65	58,39	4,067145	26	40	
66	77,63	4,351954	33	34	
68	81,48	4,400358	34	34	
69	172,6	5,150977	31	39 M	auch 171,68 und 173,63 s
70	189,63	5,245075	35	36	
71	184,55	5,217920	30	41	
71	188,63	5,239787	31	40	
71	175,18	5,165814	36	36	
71	143,66	4,967449	33 M	39 M	auch 145,31 s
72	263,93	5,575684	36	36	
72	202,29	5,309702	36	37	
72	143,42	4,965777	36	37	
72	171,47	5,144408	33 M	40	auch 181,05 und 173,73 s
73	327,01	5,789991	37	37	
73	339,89	5,828622	33 M	41	auch 340,75 und 339,75 s
74	313,4	5,747480	36	39	
75	440,86	6,088727	37	39 M	
75	609,14	6,412048	36	40	
75	518,56	6,251056	37	39	
76	905,3	6,80827	37	40	auch 912,72 s
77	1285,47	7,158880	39	39	
78	838,49	6,731603	39	40 M	
79	>1800		40	40	Abbruch

Tabelle 1:
Ergebnis-Liste (Rechnung mit *SAGE/PARI*);
Extrakt aus Tabelle 4 (siehe LOG-IN-Service).

l(*n*): dezimale Ziffernzahl von *n*
t(*n*): CPU-Zeit der Faktorisierung von *n* in Sekunden (s)
M: Mersenne-Zahl
weitere Anmerkungen: alternative Zeit bei der gleichen Aufgabe

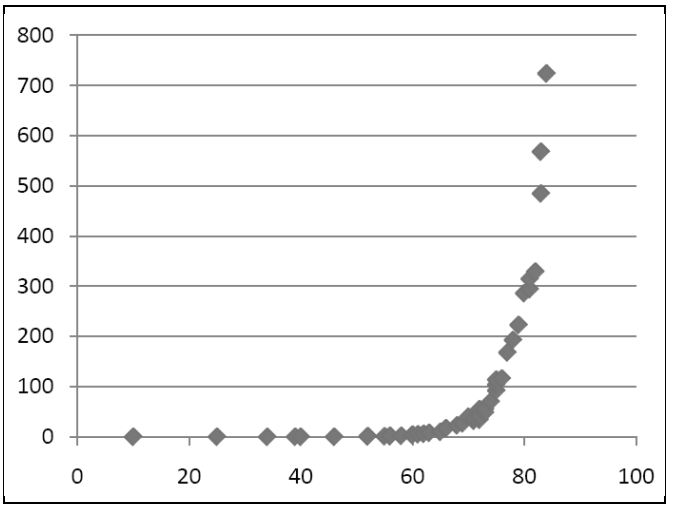

Bild 9: Faktorisierungszeit *t* in Abhängigkeit von der Zeichenlange von *n* (Rechnung mit *CrypTool 2* auf eigenem PC).

sich aber bei Berechnungen mit *CrypTool 2*: Nicht nur wird die Hardware besser genutzt (z.B. durch Verwendung beider Kerne), sondern es stehen auch leistungsfähigere Algorithmen zur Verfügung wie das *Multiple Polynomial Quadratic Sieve* MPQS (vgl. Wikipedia, Stichwort „Quadratisches Sieb"). Allerdings ist bei *CrypTool* noch nicht das Zahlkörpersieb implementiert.

Zwei besondere Zahlen

Von den Zerlegungen wollen wir hier kurz zwei besonders hervorheben. Die eine betrifft die 10-stellige Zahl 8616460799, die Zahl des englischen Ökonomen

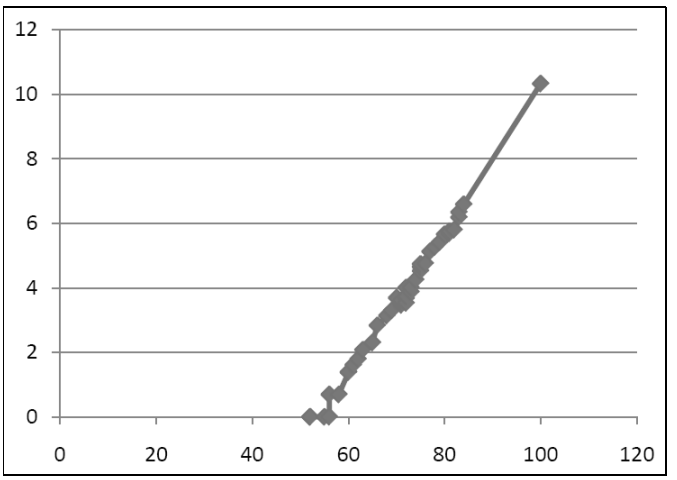

Bild 10: ln(*t*) in Abhängigkeit von der Zeichenlänge von *n* (Rechnung mit *CrypTool 2* auf eigenem PC).

und Logikers William Stanley Jevons (1835–1885). Dieser hatte in seinem 1874 erschienen Werk *The Principles of Science – A Treatise on Logic and Scientific Method* schon den möglichen Einwegcharakter der Produktbildung zweier großer Primzahlen – einer der

Fortsetzung nächste Seite

l(*n*)	*n* (Näherungswert)	*T*(*n*)	*t*(*n*)	ln(*t*(*n*))	Methode CT1
10	8616460799	0,047	0,016	–4,135167	Pollard
25	5,01856E+24	0,25	0,016	–4,135167	Pollard
34	1,24082E+33	0,187	0,016	–4,135167	Pollard
39	8,74298E+38	4,141	0,859	–0,151986	QS
40	2,17203E+39	3,686	0,766	–0,266573	QS
46	1,42725E+45	0,047	0,031	–3,473768	Williams
52	1,60003E+51	52,921	11,702	2,4597598	QS
52	4,35604E+51	39,984	10,563	2,3573573	QS
55	9,01475E+54	107	23,734	3,1669086	QS
56	1,60003E+55	78	20,155	3,0034524	QS
56	9,01475E+55	129	31,312	3,4440014	QS
58	1,35627E+57	76	28,094	3,335556	QS
60	1,00434E+59	261	61	4,1108739	QS
60	2,67620E+59	315	76	4,3307333	QS
61	2,62597E+60	426	95	4,5538769	QS
62	2,67426E+61	475	119	4,7791235	QS
63	8,53380E+62	0,25	0,031	–3,473768	Pollard
65	5,73858E+64	807	206	5,3278762	QS
66	7,88323E+65	1673	473	6,1590954	QS
68	1,93196E+67	1766	496	6,2065759	QS
69	8,94834E+68	0,156	0,032	–3,442019	Pollard
70	8,69565E+69	3498	1045	6,9517722	QS
71	1,11111E+70	3455	801	6,6858609	QS
71	1,53041E+70		1439	7,2717037	QS
71	2,28790E+70		1235	7,1188262	QS
71	2,76070E+70		1020	6,9275579	QS

Tabelle 2 (oben): Ergebnis-Liste (Experimente mit *CrypTool 1* auf eigenem PC); Extrakt aus Tabelle 4 (siehe LOG-IN-Service).

l(*n*): dezimale Ziffernzahl von *n*
T(*n*): CPU-Zeit der Faktorisierung von *n* in Sekunden bei allen angeschalteten Algorithmen
t(*n*): CPU-Zeit der Faktorisierung von *n* in Sekunden bei Wahl des angegebenen Algorithmus
Pollard: Pollards (p–1)-Verfahren
QS: Quadratisches Sieb
Williams: Williams (p+1)-Verfahren
Ausstattung des eigenen PC: 2 GByte Hauptspeicher, Intel Core 2 CPU und 2,4 GHertz Taktfrequenz

l(*n*)	*n* (Näherungswert)	*t*(n) [Sekunden]	ln(*t*(*n*))
10	8616460799	0	
25	5,01856E+24	0	
34	1,24082E+33	0	
39	8,74298E+38	0	
40	2,17203E+39	0	
46	1,42725E+45	0	
52	1,60003E+51	1	0
52	4,35604E+51	1	0
55	9,01475E+54	1	0
56	1,60003E+55	1	0
56	9,01475E+55	2	0,693147181
58	1,35627E+57	2	0,693147181
60	1,00434E+59	4	1,386294361
60	2,67620E+59	4	1,386294361
61	2,62597E+60	5	1,609437912
62	2,67426E+61	6	1,791759469
63	8,53380E+62	8	2,079441542
65	5,73858E+64	10	2,302585093
66	7,88323E+65	17	2,833213344
68	1,93196E+67	23	3,135494216
69	8,94834E+68	27	3,295836866
70	8,69565E+69	40	3,688879454
71	1,11111E+70	32	3,465735903
71	1,53041E+70	36	3,583518938
71	2,28790E+70	34	3,526360525
71	2,76070E+70	37	3,610917913
72	1E+71	39	3,663561646
72	1,82405E+71	55	4,007333185
72	2E+71	34	3,526360525
72	4,72154E+71	42	3,737669618
73	1,36986E+72	48	3,871201011
73	7,46308E+72	55	4,007333185
74	4,98736E+73	70	4,248495242
75	2,33070E+74	92	4,521788577
75	3,87467E+74	104	4,644390899
75	5,13083E+74	114	4,736198448
76	3,98613E+75	116	4,753590191
77	8,75598E+76	168	5,123963979
78	4,95090E+77	193	5,262690189
79	5,84788E+78	223	5,407171771
80	5,43974E+79	286	5,655991811
81	4,40514E+80	294	5,683579767
81	8,58080E+80	315	5,752572639
82	6,94879E+81	329	5,796057751
83	3,10024E+82	485	6,184148891
83	7,09936E+82	568	6,342121419
84	3,16742E+83	724	6,584791392
100	1,522605E+99	30405	10,32236235

Tabelle 3 (rechts): Ergebnis-Liste (Experimente mit *CrypTool 2* auf eigenem PC); Extrakt aus Tabelle 4 (siehe LOG-IN-Service).

l(*n*): dezimale Ziffernzahl von *n*
t(*n*): Wall-Zeit (End-Zeit minus Start-Zeit) der Faktorisierung

Schlüsseltatsachen beim RSA-Kryptosystem – bemerkt und geschrieben (hier aus dem Englischen übersetzt; vgl. Jevons, 1874, S. 141):

Kann der Leser sagen, welche zwei Zahlen miteinander multipliziert die Zahl 8616460799 ergeben? Ich denke, es ist unwahrscheinlich, dass das irgendjemand außer mir selbst je wissen wird.

Hier irrte Jevons gewaltig (eine Geschichte, die Józef Dénes gemäß Golomb, 1996, ausgraben konnte): Schon 1903 konnte Derrick Norman Lehmer (1867–1938) die Faktoren angeben, und Solomon Wolf Golomb (geb. 1932) zeigte in einem Artikel (Golomb, 1996), dass schon Zeitgenossen von Jevons mit den damaligen Hilfsmitteln die Zerlegung in wenigen Stunden, wahrscheinlich schon innerhalb einer Stunde, hätten leisten können. Auch Günter Ziegler erzählt die Geschichte von Jevons' Zahl in seinem hübschen Buch *Darf ich Zahlen* (Ziegler, ^{3}2010, S. 40 f.). Mit *SAGE* oder *CrypTool* gelingt die Zerlegung in weniger als 0,01 Sekunden.

Die zweite Zahl, die wir hier hervorheben wollen, ist die Zahl RSA-100 aus der *RSA-Challange* (vgl. z. B. Wikipedia, Stichwort „RSA-100"), und zwar
1522605027922533360535618378132637429718068114961380688657908494580122963258952897654000350692006139,
eine Zahl mit 100 Dezimalstellen. Sie wurde schon 1991 von Arjen K. Lenstra mit dem MPQS in wenigen Tagen zerlegt. Erstaunlicherweise benötigt CT2 nur 8 Stunden und 25 Minuten zur Zerlegung (siehe Bild 4, Seite 109). Damit bietet sich durch Parallelschaltung von Rechnern die Möglichkeit, RSA-100 innerhalb einer Unterrichtsstunde zu faktorisieren.

Beachtlich ist also, welchen Fortschritt es bei Hard- und Software gegeben hat und wie sehr selbst Ron Rivest 1977 daneben lag, als er den Zeitbedarf für die Faktorisierung einer Zahl mit 125 dezimalen Stellen (selbst unter der Annahme, dass eine modulare Multiplikation in einer Nanosekunde ausgeführt werden kann) auf $4 \cdot 10^{16}$ Jahre schätzte. RSA-129 hielt er für praktisch unzerlegbar, was Derek Atkins, Michael Graff, Arjen K. Lenstra und Paul Leylan 1994 widerlegen konnten. Auch auf privatem PC, so die Schätzung von *CrypTool 2* (diesmal Version 2.0.2093.1), liegt die zur Zerlegung von RSA-129 benötigte Zeit nur bei knapp 3 Monaten (genauer: in 83 Tagen, 18 Stunden und 55 Minuten), bei Parallelisierung entsprechend weniger. Was Rivest also nicht berücksichtigt hatte, war die Möglichkeit des Fortschritts bei der Entwicklung von Faktorisierungsalgorithmen. Da man das Alter der Erde und unseres Sonnensystems auf $4{,}55 \cdot 10^{9}$ Jahre schätzt (aktuell etwas niedriger, aber immer noch auf mindestens 4,44 Milliarden Jahre), wird die Diskrepanz der Zeitschätzung von Rivest zu der tatsächlich benötigten Zeit von *CrypTool 2* besonders anschaulich.

Fazit

Unsere Experimente zeigen, dass die Zerlegungszeiten von Semiprimzahlen bei zunehmender Länge dieser Zahlen (bei gleichem Rechner und festem Algorithmus) ungefähr exponentiell ansteigen.

Infolge schnellerer Hardware und verbesserter Algorithmen kann man auf Einzelplatz-PCs Semiprimzahlen mit einer Länge bis ca. 90 Dezimalziffern im Stundenbereich in ihre Primfaktoren zerlegen. Das aktuell schnellste Software-Werkzeug ist dabei *CrypTool 2*; im Jahr 1996 konnten wir eine 21-stellige Semiprimzahl mit 11-stelligen Faktoren in etwa 229 Sekunden zerlegen (mit dem Programm *DERIVE* 2.08 auf Compaq 386/20e, siehe Schulz, 1996); heute wird die Rechenzeit mit (abgerundeten) 0 Sekunden angegeben. Und die 45-stellige Mersenne-Semiprimzahl M_{149} konnten wir damals nicht mehr in angemessener Zeit faktorisieren; heute dagegen in unter 2,5 Sekunden (z. B. mit dem Quadratischen Sieb in *CrypTool 1*). Teilweise (z. B. bei ca. 25-stelligen Zahlen) sind wir heute – knapp 15 Jahre später – um 20 000-mal schneller.

Die heute aktuell eingesetzten Verfahren (bei SSL etc.) nutzen aber Semiprimzahlen mit wesentlich mehr als 200 Dezimalstellen und liegen damit noch weit außerhalb der Reichweite von Einzel-PCs.

Prof. Dr. Ralph-Hardo Schulz
Freie Universität Berlin
Fachbereich Mathematik und Informatik
Institut für Mathematik
Arnimallee 3
14195 Berlin

E-Mail: schulz@math.fu-berlin.de

Helmut Witten
Brandenburgische Straße 23
10707 Berlin

E-Mail: helmut@witten-berlin.de

Bernhard Esslinger (Universität Siegen und CrypTool) und Malte Hornung (Freie Universität Berlin) danken wir herzlich für wertvolle Hinweise bei der Entstehung dieses Artikels. Unser Dank gilt auch Sven Rech vom Lehrstuhl für Verteilte Systeme der Universität Duisburg-Essen für technische Auskünfte zu *CrypTool 2*, insbesondere zur Bibliothek *msieve*.

Über den **LOG-IN-Service** (siehe Seite 143) ist die ausführliche Tabelle 4 der Ergebnisse zu den Zeit-Experimenten zu erhalten.

Literatur und Internetquellen

BSI – Bundesamt für Informationstechnik: IT-Grundschutz-Kataloge M 2.164 – Auswahl eines geeigneten kryptographischen Verfahrens. Stand: 2009.
https://www.bsi.bund.de/cln_183/ContentBSI/grundschutz/kataloge/m/m02/m02164.html

CrypTool: Lernprogramm für Kryptographie und Kryptoanalyse.
http://www.cryptool.org/
CT1 – CrypTool Version 1.4.30 vom 4. August 2010.
http://www.cryptool.org/index.php/de/download-topmenu-63.html
CT2 – CrypTool Version 2.0 (Beta-Version).
http://cryptool2.vs.uni-due.de/

Esslinger, B.; Koy, H.: Kryptologie im Unterricht mit CrypTool. In: LOG IN, 29. Jg. (2009), Heft 157/158, S. 75–78.

Esslinger, B. u. a.: Das CrypTool-Skript – Kryptographie, Mathematik und mehr. Hintergrundmaterial und Zusatzinformationen zum freien E-Learning-Programm CrypTool (mit Code-Beispielen zur Zahlentheorie, geschrieben in Sage), veröffentlicht mit CrypTool-Version 1.4.30. Frankfurt a. M.: 2010.
http://www.cryptool.de/download/CrypToolScript-de.pdf

Golomb, S. W.: On factoring Jevons' number. In: Cryptologia, 20. Jg. (1996), Heft 3, S. 243–246.

Jevons, W. St.: The Principles of Science – A Treatise on Logic and Scientific Method. Volume I. London und New York: Macmillan, 1874.
http://www.archive.org/stream/principlesscien02jevogoog#page/n3/mode/2up

Kleinjung, Th. u. a.: Factorization of a 768-bit RSA modulus – version 1.4, February 18, 2010.
http://eprint.iacr.org/2010/006.pdf

msieve – Multiple Polynomial Quadratic Sieve:
http://sourceforge.net/projects/msieve/

Ribenboim, P.: The New Book of Prime Number Records. Berlin; Heidelberg; New York: Springer, ^{3}1996.

RSA Laboratories: The RSA Challenge Numbers. 2010.
http://www.rsa.com/rsalabs/node.asp?id=2093

SAGE – Version 4.5 vom 16. Juli 2010 (aktuelle Version: 4.6.1 vom 13. Januar 2011):
http://www.sagemath.org/

Schulz, R.-H.: Primfaktorzerlegung – Experimente zum Zeitaufwand. In: LOG IN, 16. Jg. (1996), Heft 5/6, S. 22–26.

Schulz, R.-H.: Codierungstheorie – Eine Einführung. Wiesbaden: Vieweg, ^{2}2003 (insbesondere: Kapitel IV).

Wikipedia – Stichwort „Pollard-p-1-Methode":
http://de.wikipedia.org/wiki/Pollard-p-1-Methode

Wikipedia – Stichwort „Pollard-Rho-Methode":
http://de.wikipedia.org/wiki/Pollard-Rho-Methode

Wikipedia – Stichwort „Quadratisches Sieb":
http://de.wikipedia.org/wiki/Quadratisches_Sieb

Wikipedia – Stichwort „RSA-100":
http://en.wikipedia.org/wiki/RSA_numbers#RSA-100

Wikipedia – Stichwort „Zahlkörpersieb":
http://de.wikipedia.org/wiki/Zahlk%C3%B6rpersieb

Witten, H.; Schulz, R.-H.: RSA & Co. in der Schule – Moderne Kryptologie, alte Mathematik, raffinierte Protokolle. Neue Folge – Teil 1: RSA für Einsteiger. In: LOG IN, 26. Jg. (2006a), Heft 140, S. 45–54.

Witten, H.; Schulz, R.-H.: RSA & Co. in der Schule – Moderne Kryptologie, alte Mathematik, raffinierte Protokolle. Neue Folge – Teil 2: RSA für große Zahlen. In: LOG IN, 26. Jg. (2006b), Heft 143, S. 50–58.

Witten, H.; Schulz, R.-H.: RSA & Co. in der Schule – Moderne Kryptologie, alte Mathematik, raffinierte Protokolle. Neue Folge – Teil 3: RSA und die elementare Zahlentheorie. In: LOG IN, 28. Jg. (2008), Heft 152, S. 60–70.

Witten, H.; Schulz, R.-H.: RSA & Co. in der Schule – Moderne Kryptologie, alte Mathematik, raffinierte Protokolle. Neue Folge – Teil 4: Gibt es genügend Primzahlen für RSA? In: LOG IN, 30. Jg. (2010), Heft 163/164, S. 97–103.

Ziegler, G. M.: Darf ich Zahlen? Geschichten aus der Mathematik. München: Piper, ^{3}2010.

Alle Internetquellen wurden zuletzt am 30. Dezember 2010 geprüft.