Primzahltests und Faktorisierung

Christian Ostermeier,
Marcel Goehring,
Franziska Göbel

07.12.2006

Inhaltsverzeichnis

Inhaltsverzeichnis zu den probabilistischen Primzahltests

Notwendigkeit von (großen) Primzahlen

- RSA ist asymmetrisches Verschlüsselungsverfahren
- Sender sendet Nachricht verschlüsselt mit dem öffentlichen Schlüssel des Empfängers
- Empfänger kann das entschlüsseln, weil er die Zusammensetzung des öffentlichen Schlüssels kennt
- Für diese Zusammensetzung braucht man zwei Primzahlen (aus Sicherheitsgründen etwa gleicher Länge, nicht aber gleicher Größe)

- **Formeln zur Generierung von Primzahlen**?
- **Nein**, weil die bekannten Formeln nur ungeeignete Zahlen ausgeben, die auch nur relativ sicher Primzahlen sind und nochmals auf ihre Primzahleigenschaft getestet werden müssten
- Zufälligkeit der Primzahlen fraglich (Stichworte: Mersenne-Primzahlen, Fermatsche Primzahlen)

Vorgehensweise zur Erzeugung passender Primzahlen

- Jeweils Zahl gewünschter Länge zufällig generieren und dann Primalität testen
  - Aber muss man nicht viel zu viele Zahlen generieren und testen, um letztlich eine Primzahl zu erhalten?
  - Und ist der Algorithmus so schnell, das in annehmbarer Zeit zu schaffen?
  - Sind die Primzahlen zufällig oder systematisch in $\mathbb{N}$ verteilt?
    - Menge der Primzahlen relativ dicht
    - mit zunehmender Größe wird die Voraussage eines mittleren Abstandes zur nächsten Primzahl zunehmend unsicherer $\Rightarrow$ Primzahlen nicht systematisch verteilt, Verteilung wird immer unregelmäßiger (vermutlich unendl. viele Primzahlzwillinge!)

Prime Number Theorem (Gauß)

Gibt $\pi(N)$ die Anzahl der Primzahlen, die kleiner als N sind, an, so gilt: $\pi(N) \approx \frac{N}{lnN}$

Wahrscheinlichkeit, dass zufällige 512 bit-Zahl prim ist:
$\frac{1}{ln2^{512}} \approx \frac{1}{355}$

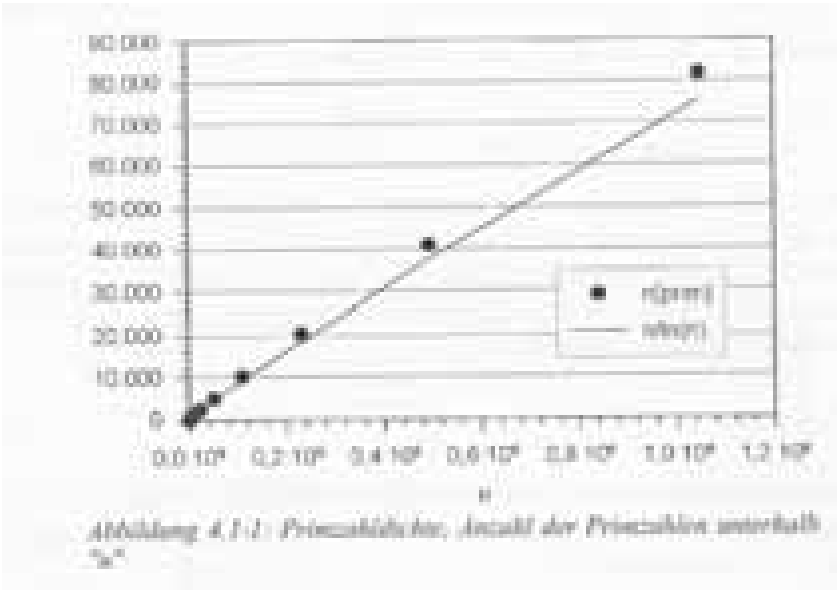

Abbildung: aus „Verschlüsselungsalgorithmen“ von Gilbert Brands

Einfache Alg. für den Primzahltest einer Zahl n

- **Probedivision**
- **Sieb des Eratosthenes**
  - Liefert alle Primzahlen von 1 bis N in einer Liste (N durch nötigen Rechenaufwand begrenzt)
- Anwendung: bei Primzahltest-/Faktorisierungsalg. wird vorausgehend zu einem der später vorgestellten Verfahren die Probedivision anhand einer solchen Primzahltabelle bis üblicherweise 10^6 durchgeführt (so findet man schnell einen kleinen Teiler, wenn vorhanden)

Der Fermat-Test

Der Kleine Satz von Fermat

Ist n eine Primzahl, so gilt $a^{n-1} \equiv 1 \pmod{n}$ für alle $a \in \mathbb{Z}$ mit $\gcd(a, n) = 1$.

- äquivalente Aussage: $a^n \equiv a \pmod{n}$ für jedes $a \in \mathbb{Z}$
- Die Kongruenz gilt bei jeder Primzahl für jedes a
- Wählt man a (zw. 1 und n) zufällig, rechnet damit den linken Term aus und erhält einen von 1 verschiedenen Wert, weiß man, dass es sich um keine Primzahl handelt
- Oben keine „genau dann, wenn“-Beziehung $\Rightarrow$ bei einem Ergebnis von 1 kann die Zahl Primzahl sein, muss aber nicht
- Liefert im Gegensatz zur fortgeführten Probedivision keine Faktorisierung der Zahl

Pseudoprim- und Carmichael-Zahlen

Der Kleine Satz von Fermat

Ist n eine Primzahl, so gilt $a^{n-1} \equiv 1 \pmod{n}$ für alle $a \in \mathbb{Z}$ mit $\gcd(a, n) = 1$.

- Gilt obige Kongruenz für eine zusammengesetzte („composite“) Zahl n und *eine* ganze Zahl a, dann heißt n (frematsche) **Pseudoprimzahl** zur Basis a
  - Bsp.: $341 = 11 \cdot 31$ ist Pseudoprimzahl zur Basis 2, weil $2^{340} = 1 \pmod{341}$
- Ist n eine Pseudoprimzahl zur Basis a für *alle* ganzen Zahlen a mit $\gcd(a, n) = 1$, dann heißt n **Carmichael-Zahl**
  - Bsp. für Carmichael-Zahl: $561 = 3 \cdot 11 \cdot 17$
  - es existieren unendlich viele Carmichael-Zahlen - bei allen kann der (hier beschriebene) Fermat-Test die Zusammengesetztheit nicht feststellen

Monte-Carlo-Algorithmus und Co.

- *Entscheidungsproblem: Antwort nur „ja“ oder „nein“*
- *Probabilistischer Algorithmus: benutzt Zufallszahlen (im Gegensatz zum deterministischen Algorithmus), Bsp.: Fermat-Test*
- *Yes-biased („ja“-fixiert) Monte-Carlo-Algorithmus: probabilistischer Alg. für ein Entscheidungsproblem, bei dem die „ja“-Antwort immer korrekt ist, die „nein“-Antwort aber falsch sein kann (eine Antwort wird immer geliefert)*
- *No-biased Monte-Carlo-Alg.: wie oben, nur ist hier die „nein“-Antwort immer korrekt, während die „ja“-Antwort falsch sein kann*
- *Yes-biased Monte-Carlo-Alg. hat Fehlerwahrscheinlichkeit von ϵ, wenn die Wahrscheinlichkeit, dass der Alg. fälschlicherweise mit „nein“ antwortet, bei höchstens ϵ liegt*
- *Las-Vegas-Alg. liefert nicht immer eine Antwort, aber wenn, dann ist diese Antwort auch korrekt*

Solovay-Strassen-Alg. (1977, Robert Solovay, Volker Strassen)

choose a random integer a such that $1 \leq a \leq n-1$

$x \leftarrow \left(\frac{a}{n}\right)$

if $x = 0$

then return („n is composite“)

$y \leftarrow a^{(n-1)/2} \pmod{n}$

if $x \equiv y \pmod{n}$

then return („n is prime“)

else return („n is composite“)

(aus „Cryptography - Theory and Practice - Second Edition“ von Douglas R. Stinson)

Quadratischer Rest

Definition

Sei p eine ungerade Primzahl und a eine ganze Zahl. Weiter gelte $a \not\equiv 0 \pmod{p}$. a heißt quadratischer Rest („quadratic residue“) modulo p, wenn die Kongruenz $y^2 \equiv a \pmod{p}$ eine Lösung $y \in \mathbb{Z}_p$ hat. Existiert keine solche Lösung, heißt a nicht-quadratischer Rest („quadratic non-residue“).

Bsp.: $\mathbb{Z}_{11}$ (mod-Tabelle)

									...	-12
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
0	1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20	21
22	...									

Bsp. zu quadratischen Resten

1^2	$= 1$
2^2	$= 4$
3^2	$= 9$
4^2	$= 5$
5^2	$= 3$
6^2	$= 3$
7^2	$= 5$
8^2	$= 9$
9^2	$= 4$
10^2	$= 1$

									...	-12
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
0	1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20	21
22	...									

- Werte rechts der Gleichheitszeichen sind quadratische Reste modulo 11

Eulers Satz

Theorem

Sei p eine ungerade Primzahl. a ist genau dann ein quadratischer Rest („quadratic residue“) modulo p, wenn $a^{(p-1)/2} \equiv 1 \pmod{p}$.

- Theorem ermöglicht effiziente Beantwortung der Frage „ist a quadratischer Rest modulo p?“
  - Nutzung des Square-and-Multiply-Alg., $O((log\ p)^3)$

Das Legendre-Symbol

Definition

Sei p eine ungerade Primzahl. Für jede ganze Zahl a ist das Legendre-Symbol $\left(\frac{a}{p}\right)$ wie folgt definiert:

$$\left(\frac{a}{p}\right) = \begin{cases} 0, & \text{wenn } a \equiv 0 \pmod{p} \\ 1, & \text{wenn a quadratischer Rest modulo p} \\ -1, & \text{wenn a nicht-quadratischer Rest modulo p} \end{cases}$$

Legendre-Symbole effizient auswerten

Theorem

Ist p eine ungerade Primzahl, so gilt $\left(\frac{a}{p}\right) \equiv a^{(p-1)/2} \pmod{p}$.

effiziente Auswertung des Legendre-Symbols

Damit ist

$$\left(\frac{a}{p}\right) \equiv a^{(p-1)/2} \pmod{p}$$

$$= \begin{cases} 0, & \text{wenn } a \equiv 0 \pmod{p} \\ 1, & \text{wenn a quadratischer Rest modulo p} \\ -1, & \text{wenn a nicht-quadratischer Rest modulo p} \end{cases}$$

Das Jacobi-Symbol als Erweiterung auf zusammengesetzte Zahlen

Definition

Sei n eine ungerade, positive ganze Zahl (nicht wie eben zwingend prim!) mit der Primfaktorzerlegung $n = \prod_{i=1}^{k} p_i^{e_i}$ und $a \in \mathbb{Z}$, dann ist das Jacobi-Symbol $\left(\frac{a}{n}\right)$ definiert als $\left(\frac{a}{n}\right) = \prod_{i=1}^{k} \left(\frac{a}{p_i}\right)^{e_i}$.

Jacobi-Symbol-Beispiel

Betrachten wir das Jacobi-Symbol $\left(\frac{6278}{9975}\right)$. Die kanonische Primzahlzerlegung von 9975 ist $9975 = 3 \cdot 5^2 \cdot 7 \cdot 19$. Damit ist $\left(\frac{6278}{9975}\right) = \left(\frac{6278}{3}\right)\left(\frac{6278}{5}\right)^2\left(\frac{6278}{7}\right)\left(\frac{6278}{19}\right) = \left(\frac{2}{3}\right)\left(\frac{3}{5}\right)^2\left(\frac{6}{7}\right)\left(\frac{8}{19}\right) = (-1)(-1)^2(-1)(-1) = -1$.

Jacobi-Symbol ohne Primfaktorzerlegung auswerten (1)

Regel 1

Ist n eine ungerade nat. Zahl und es gilt $m_1 \equiv m_2 \pmod{n}$, dann gilt $\left(\frac{m_1}{n}\right) = \left(\frac{m_2}{n}\right)$.

Regel 2

Ist n eine ungerade nat. Zahl, so gilt

$$\left(\frac{2}{n}\right) = \begin{cases} 1, & \text{wenn } n \equiv \pm 1 \pmod{8} \\ -1, & \text{wenn } n \equiv \pm 3 \pmod{8} \end{cases}$$

Jacobi-Symbol ohne Primfaktorzerlegung auswerten (2)

Regel 3

Ist n eine ungerade nat. Zahl, so gilt $\left(\frac{m_1 m_2}{n}\right) = \left(\frac{m_1}{n}\right)\left(\frac{m_2}{n}\right)$.

Insbesondere gilt für $m = 2^k t$ mit ungeradem t $\left(\frac{m}{n}\right) = \left(\frac{2}{n}\right)^k \left(\frac{t}{n}\right)$.

Regel 4

Sind m und n ungerade nat. Zahlen, so gilt

$$\left(\frac{m}{n}\right) = \begin{cases} -\left(\frac{n}{m}\right), & \text{wenn } m \equiv n \equiv 3 \pmod 4 \\ \left(\frac{n}{m}\right), & \text{sonst} \end{cases}$$

Solovay-Strassen-Alg. (1977, Robert Solovay, Volker Strassen)

choose a random integer a such that $1 \leq a \leq n-1$

$x \leftarrow \left(\frac{a}{n}\right)$

if $x = 0$

then return („n is composite“)

$y \leftarrow a^{(n-1)/2} \pmod{n}$

if $x \equiv y \pmod{n}$

then return („n is prime“)

else return („n is composite“)

(aus „Cryptography - Theory and Practice - Second Edition“ von Douglas R. Stinson)

Laufzeit und Fehlerabschätzung

- Solovay-Strassen-Alg. hat *polynomielle Laufzeit* (genau: $O((logn)^3)$, weil die Komponenten alle effizient berechenbar sind
- Ergebnis kein Beweis, wenn behauptet wird, dass es sich um eine Primzahl handelt $\Rightarrow$ Wiederholung nötig, um *Fehler beliebig klein* zu kriegen
- Solovay-Strassen ist yes-biased Monte Carlo Alg. für die Frage „ist die nat. Zahl n zusammengesetzt?“, d.h. antwortet der Alg. (einmaliger Durchlauf) mit „ja“, so ist das garantiert richtig, antwortet er mit „nein“, ist die Zahl mit der Wahrscheinlichkeit $\epsilon < \frac{1}{2}$ zusammengesetzt
  - analog zu Fermatschen Pseudoprimzahlen gibt es hier *Eulersche Pseudoprimzahlen*, nämlich diejenigen zusammengesetzten n, für die $\left(\frac{a}{n}\right) \equiv a^{(n-1)/2} \pmod{p}$ gilt
  - man kann zeigen, dass n eine *Eulersche Pseudoprimzahl zu höchstens der Hälfte der möglichen a* ist

  $\Rightarrow$ praxistauglich

Weiterführendes

- Miller-Rabin-Test
  - prinzipiell ähnlich dem Solovay-Strassen-Alg. (Zahlentheorie!)
  - ebenfalls yes-biased Monte Carlo Alg.
  - nutzt einen Satz von Miller, der den kleinen Satz von Fermat hinsichtlich Carmichael-Zahlen optimiert
  - $\epsilon < \frac{1}{4}$, d.h. (Fehler-)Wahrscheinlichkeit bei einmaligem Durchlauf für das Ergebnis „n ist prim“, obwohl n zusammengesetzt, $< \frac{1}{4}$
  - Komplexität ebenfalls $O((logn)^3$, in der Praxis aber schneller als Solovay-Strassen
- Elliptic Curve Primality Proving (ECPP)
  - probabilistischer Las-Vegas-Alg.
- AKS
  - deterministischer Alg. mit polynomieller Laufzeit

Deterministische Primzahltests

- Vorteil:
  - Liefern 100%-ig korrektes Ergebnis, ob n eine Primzahl ist, oder nicht
- Nachteil:
  - Langsamer als probabilistische Verfahren

Welche Verfahren gibt es?

- Verschiedene Brute-Force-Varianten (Probedivision)
- Sieb des Eratosthenes
- Lucas-Lehmer-Test für Mersennesche Zahlen
- Lucas-Lehmer-Test allgemein
- AKS-Methode
- Jacobi-Summen-Verfahren

Das Sieb des Eratosthenes

- Eratosthenes von Kyrene, griechicher Gelehrter, Kyrene, 3. JH. v. Chr.
- Kein Primzahltest im eigentlichen Sinne, sondern Bestimmung aller Primzahlen unterhalb einer Schranke N
- Findet auch heute noch Anwendung zur Aufstellung von Primzahltafeln

Das Sieb des Eratosthenes

- Schreibe alle Zahlen von 2 bis N auf
- Streiche für jede Zahl $2 \leq k \leq \sqrt{N}$ alle Nachfolger, die ein Vielfaches von k sind
- Alle nicht gestrichenen Zahlen sind alle Primzahlen, die kleiner oder gleich N sind

Das Sieb des Eratosthenes

- Schreibe alle Zahlen von 2 bis N auf
- Streiche für jede Zahl $2 \leq k \leq \sqrt{N}$ alle Nachfolger, die ein Vielfaches von k sind
- Alle nicht gestrichenen Zahlen sind alle Primzahlen, die kleiner oder gleich N sind

Beispiel:
N = 100

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

Das Sieb des Eratosthenes

- Schreibe alle Zahlen von 2 bis N auf
- Streiche für jede Zahl $2 \leq k \leq \sqrt{N}$ alle Nachfolger, die ein Vielfaches von k sind
- Alle nicht gestrichenen Zahlen sind alle Primzahlen, die kleiner oder gleich N sind

Beispiel:
N = 100

	2	3	~~4~~	5	~~6~~	7	~~8~~	9	~~10~~
11	~~12~~	13	~~14~~	15	~~16~~	17	~~18~~	19	~~20~~
21	~~22~~	23	~~24~~	25	~~26~~	27	~~28~~	29	~~30~~
31	~~32~~	33	~~34~~	35	~~36~~	37	~~38~~	39	~~40~~
41	~~42~~	43	~~44~~	45	~~46~~	47	~~48~~	49	~~50~~
51	~~52~~	53	~~54~~	55	~~56~~	57	~~58~~	59	~~60~~
61	~~62~~	63	~~64~~	65	~~66~~	67	~~68~~	69	~~70~~
71	~~72~~	73	~~74~~	75	~~76~~	77	~~78~~	79	~~80~~
81	~~82~~	83	~~84~~	85	~~86~~	87	~~88~~	89	~~90~~
91	~~92~~	93	~~94~~	95	~~96~~	97	~~98~~	99	~~100~~

Das Sieb des Eratosthenes

- Schreibe alle Zahlen von 2 bis N auf
- Streiche für jede Zahl $2 \leq k \leq \sqrt{N}$ alle Nachfolger, die ein Vielfaches von k sind
- Alle nicht gestrichenen Zahlen sind alle Primzahlen, die kleiner oder gleich N sind

Beispiel:
N = 100

	2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~
11	~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19	~~20~~
~~21~~	~~22~~	23	~~24~~	25	~~26~~	~~27~~	~~28~~	29	~~30~~
31	~~32~~	~~33~~	~~34~~	35	~~36~~	37	~~38~~	~~39~~	~~40~~
41	~~42~~	43	~~44~~	~~45~~	~~46~~	47	~~48~~	49	~~50~~
~~51~~	~~52~~	53	~~54~~	55	~~56~~	~~57~~	~~58~~	59	~~60~~
61	~~62~~	~~63~~	~~64~~	65	~~66~~	67	~~68~~	~~69~~	~~70~~
71	~~72~~	73	~~74~~	~~75~~	~~76~~	77	~~78~~	79	~~80~~
~~81~~	~~82~~	83	~~84~~	85	~~86~~	~~87~~	~~88~~	89	~~90~~
91	~~92~~	~~93~~	~~94~~	95	~~96~~	97	~~98~~	~~99~~	~~100~~

Das Sieb des Eratosthenes

- Schreibe alle Zahlen von 2 bis N auf
- Streiche für jede Zahl $2 \leq k \leq \sqrt{N}$ alle Nachfolger, die ein Vielfaches von k sind
- Alle nicht gestrichenen Zahlen sind alle Primzahlen, die kleiner oder gleich N sind

Beispiel:
N = 100

	2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~
11	~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19	~~20~~
~~21~~	~~22~~	23	~~24~~	~~25~~	~~26~~	~~27~~	~~28~~	29	~~30~~
31	~~32~~	~~33~~	~~34~~	~~35~~	~~36~~	37	~~38~~	~~39~~	~~40~~
41	~~42~~	43	~~44~~	~~45~~	~~46~~	47	~~48~~	49	~~50~~
~~51~~	~~52~~	53	~~54~~	~~55~~	~~56~~	~~57~~	~~58~~	59	~~60~~
61	~~62~~	~~63~~	~~64~~	~~65~~	~~66~~	67	~~68~~	~~69~~	~~70~~
71	~~72~~	73	~~74~~	~~75~~	~~76~~	77	~~78~~	79	~~80~~
~~81~~	~~82~~	83	~~84~~	~~85~~	~~86~~	~~87~~	~~88~~	89	~~90~~
91	~~92~~	~~93~~	~~94~~	~~95~~	~~96~~	97	~~98~~	~~99~~	~~100~~

Das Sieb des Eratosthenes

- Schreibe alle Zahlen von 2 bis N auf
- Streiche für jede Zahl $2 \leq k \leq \sqrt{N}$ alle Nachfolger, die ein Vielfaches von k sind
- Alle nicht gestrichenen Zahlen sind alle Primzahlen, die kleiner oder gleich N sind

Beispiel:
N = 100

	2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~
11	~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19	~~20~~
~~21~~	~~22~~	23	~~24~~	~~25~~	~~26~~	~~27~~	~~28~~	29	~~30~~
31	~~32~~	~~33~~	~~34~~	~~35~~	~~36~~	37	~~38~~	~~39~~	~~40~~
41	~~42~~	43	~~44~~	~~45~~	~~46~~	47	~~48~~	~~49~~	~~50~~
~~51~~	~~52~~	53	~~54~~	~~55~~	~~56~~	~~57~~	~~58~~	59	~~60~~
61	~~62~~	~~63~~	~~64~~	~~65~~	~~66~~	67	~~68~~	~~69~~	~~70~~
71	~~72~~	73	~~74~~	~~75~~	~~76~~	~~77~~	~~78~~	79	~~80~~
~~81~~	~~82~~	83	~~84~~	~~85~~	~~86~~	~~87~~	~~88~~	89	~~90~~
~~91~~	~~92~~	~~93~~	~~94~~	~~95~~	~~96~~	97	~~98~~	~~99~~	~~100~~

Das Sieb des Eratosthenes

- Bewertung
  - Exponentielle Laufzeit
  - Hoher Platzbedarf: Alle Zahlen bis einschließlich N müssen im Speicher gehalten werden
  - Sehr aufwändig, wenn es darum geht eine einzige Zahl zu testen
  - Jedoch effizient, um alle Primzahlen eines Zahlenbereiches zu finden

Lucas-Lehmer-Test für Mersennesche Zahlen

- Marin Mersenne (Französischer Mönch und Priester 16./17.JH.)
- Mersennesche Zahlen: $M_n := 2^n - 1$
- Beispiele: $M_0 := 2^0 - 1 = 0$; $M_1 := 2^1 - 1 = 1$; $M_5 := 2^5 - 1 = 31$

Lucas-Lehmer-Test für Mersennesche Zahlen

- Marin Mersenne (Französischer Mönch und Priester 16./17.JH.)
- Mersennesche Zahlen: $M_n := 2^n - 1$
- Beispiele: $M_0 := 2^0 - 1 = 0$; $M_1 := 2^1 - 1 = 1$; $M_5 := 2^5 - 1 = 31$
- Eigenschaft: M_n Primzahl $\Rightarrow$ n Primzahl
  Damit folgt: n nicht Primzahl $\Rightarrow$ M_n nicht Primzahl
  - wähle Primzahl n
  - berechne M_n
  - prüfe mit dem Lucas-Lehmer-Test, ob M_n eine Primzahl ist

Lucas-Lehmer-Test für Mersennesche Zahlen

- Marin Mersenne (Französischer Mönch und Priester 16./17.JH.)
- Mersennesche Zahlen: $M_n := 2^n - 1$
- Beispiele: $M_0 := 2^0 - 1 = 0$; $M_1 := 2^1 - 1 = 1$; $M_5 := 2^5 - 1 = 31$
- Eigenschaft: M_n Primzahl $\Rightarrow$ n Primzahl
  Damit folgt: n nicht Primzahl $\Rightarrow$ M_n nicht Primzahl
  - wähle Primzahl n
  - berechne M_n
  - prüfe mit dem Lucas-Lehmer-Test, ob M_n eine Primzahl ist
- schnelles Erzeugen von großen Primzahlen
- M_{756839} hat im Dezimalsystem etwa 228.000 Stellen

Lucas-Lehmer-Test für Mersennesche Zahlen

- Algorithmus:
  - $s_0 := 4$
  - für $k = 0 \,..\, n-3$ berechne:
    $$s_{k+1} := s_k^2 - 2 \ mod \ M_n$$
- M_n Primzahl $\Leftrightarrow$ s_{n-2} ist durch M_n teilbar (ohne $mod \ M_n$)
- M_n Primzahl $\Leftrightarrow$ $s_{n-2} = 0$

Lucas-Lehmer-Test für Mersennesche Zahlen

- Algorithmus:
  - $s_0 := 4$
  - für $k = 0 \,..\, n-3$ berechne:
    $$s_{k+1} := s_k^2 - 2 \ mod \ M_n$$
- M_n Primzahl $\Leftrightarrow$ s_{n-2} ist durch M_n teilbar (ohne $mod \ M_n$)
- M_n Primzahl $\Leftrightarrow$ $s_{n-2} = 0$

Beispiel:

- $M_7 := 2^7 - 1 = 127$
- $s_0 := 4$
  $s_1 := (4^2 - 2) \quad mod \ 127 = 14$
  $s_2 := (14^2 - 2) \quad mod \ 127 = 67$
  $s_3 := (67^2 - 2) \quad mod \ 127 = 42$
  $s_4 := (42^2 - 2) \quad mod \ 127 = 111$
  $s_5 := (111^2 - 2) \ mod \ 127 = 0$

$\Rightarrow M_7 = 127$ ist eine Primzahl

Lucas-Lehmer-Test allgemein

- Ist n eine Primzahl?
- Voraussetzung: Primfaktorzerlegung von n-1 muss bekannt sein

Lucas-Lehmer-Test allgemein

- Ist n eine Primzahl?
- Voraussetzung: Primfaktorzerlegung von n-1 muss bekannt sein
- Algorithmus:
  - Für $a = 2 \,..\, n-1$ tue:

    Wenn $a^{n-1} \equiv 1 \ mod \ n$

    Wenn für alle Primfaktoren q von n-1 gilt $a^{(n-1)/q} \not\equiv 1 \ (mod \ n)$

    return „Primzahl“

Lucas-Lehmer-Test allgemein

- Ist n eine Primzahl?
- Voraussetzung: Primfaktorzerlegung von n-1 muss bekannt sein
- Algorithmus:
  - Für $a = 2 \,..\, n-1$ tue:

    Wenn $a^{n-1} \equiv 1 \ mod \ n$

    Wenn für alle Primfaktoren q von n-1 gilt $a^{(n-1)/q} \not\equiv 1 \ (mod \ n)$

    return „Primzahl“
- Wichtig: Verwendung schneller Exponentiationsverfahren

Lucas-Lehmer-Test allgemein

- Algorithmus:

  Für $a = 2 \,..\, n-1$ tue:

  Wenn $a^{n-1} \equiv 1 \ mod \ n$

  Wenn für alle Primfaktoren q von n-1 gilt $a^{(n-1)/q} \not\equiv 1 \ (mod \ n)$

  return „Primzahl“

- Beispiel: $n = 71$ (offensichtlich keine Mersennesche Zahl)

  Primfaktoren von $n-1$: $71 - 1 = 70 = 2 \cdot 5 \cdot 7$

  $\Rightarrow a = 11$ (kleinstmögliches a)

  $11^{35} - 1 \ mod \ 71 = 69 \neq 0$

  $11^{14} - 1 \ mod \ 71 = 53 \neq 0$

  $11^{10} - 1 \ mod \ 71 = 31 \neq 0$

  $\Rightarrow$ 71 ist eine Primzahl

Lucas-Lehmer-Test

- Bemerkungen zu den Lucas-Lehmer-Tests:
  - Nur eingeschränkt verwendbar
  - Einfach zu implementieren
  - Für Mersennesche Zahlen: sehr schnell, trotz Determinismus (polynomielle Laufzeit): $O(n^2 \log n)$

AKS - Verfahren

- Benannt nach den indischen Wissenschaftlern Agrawal, Kayal und Saxena
- 2002 vorgestellt
- Erster deterministischer Primzahltest, der in polynomieller Zeit prüft, ob eine Zahl eine Primzahl ist
- Laufzeit des ursprünglichen Algorithmus: $O((logn)^{10+\epsilon})$
- Spätere Implementationen: $O((logn)^{6+\epsilon})$; $O((logn)^{3+\epsilon}))$
- Damit wurde gezeigt, dass $PRIMES \in P$

Primzahltests - Fazit

- Es gibt deterministische und probabilistische Testverfahren
- Deterministische Testverfahren sind sicher, aber benötigen sehr viel Zeit
- Probabilistische Verfahren sind schnell und eignen sich deshalb für große Zahlen, besitzen jedoch ein „Restrisiko“
- Es gibt Zahlen, die lassen sich auch mit deterministischen Verfahren sehr schnell testen (z.B. Mersennesche Zahlen)
- Optimal sind Kombinationen verschiedenster Testverfahren

Faktorisierung - Einleitung

- Die Sicherheit von RSA hängt maßgeblich davon ab, wie effizient eine Zahl faktorisiert werden kann

Faktorisierung - Einleitung

- Die Sicherheit von RSA hängt maßgeblich davon ab, wie effizient eine Zahl faktorisiert werden kann
- Ausgangssituation: Eine Zahl n, von der bekannt ist, dass es keine Primzahl ist.

Faktorisierung - Einleitung

- Die Sicherheit von RSA hängt maßgeblich davon ab, wie effizient eine Zahl faktorisiert werden kann
- Ausgangssituation: Eine Zahl n, von der bekannt ist, dass es keine Primzahl ist.
- Begriffsklärung:
  - Zerlegen einer Zahl n in zwei Faktoren a und b ungleich Eins (Faktorzerlegung)

$$n = a \cdot b$$

Faktorisierung - Einleitung

- Die Sicherheit von RSA hängt maßgeblich davon ab, wie effizient eine Zahl faktorisiert werden kann
- Ausgangssituation: Eine Zahl n, von der bekannt ist, dass es keine Primzahl ist.
- Begriffsklärung:
  - Zerlegen einer Zahl n in zwei Faktoren a und b ungleich Eins (Faktorzerlegung)
    $$n = a \cdot b$$
  - Zerlegen einer Zahl n in ihre Primfaktoren (Primfaktorzerlegung)
    $$n = p_1^{e_1} \cdot p_2^{e_2} \cdot \ldots \cdot p_k^{e_k}$$

Faktorisierung - Einleitung

- Die Sicherheit von RSA hängt maßgeblich davon ab, wie effizient eine Zahl faktorisiert werden kann
- Ausgangssituation: Eine Zahl n, von der bekannt ist, dass es keine Primzahl ist.
- Begriffsklärung:
  - Zerlegen einer Zahl n in zwei Faktoren a und b ungleich Eins (Faktorzerlegung)
    $$n = a \cdot b$$
  - Zerlegen einer Zahl n in ihre Primfaktoren (Primfaktorzerlegung)
    $$n = p_1^{e_1} \cdot p_2^{e_2} \cdot \ldots \cdot p_k^{e_k}$$
- Beispiel: $30 = 2 \cdot 3 \cdot 5$

Faktorisierung - Einleitung

- Die Sicherheit von RSA hängt maßgeblich davon ab, wie effizient eine Zahl faktorisiert werden kann
- Ausgangssituation: Eine Zahl n, von der bekannt ist, dass es keine Primzahl ist.
- Begriffsklärung:
  - Zerlegen einer Zahl n in zwei Faktoren a und b ungleich Eins (Faktorzerlegung)
    $$n = a \cdot b$$
  - Zerlegen einer Zahl n in ihre Primfaktoren (Primfaktorzerlegung)
    $$n = p_1^{e_1} \cdot p_2^{e_2} \cdot \ldots \cdot p_k^{e_k}$$
- Beispiel: $30 = 2 \cdot 3 \cdot 5$
- Eine Primfaktorzerlegung ist immer eindeutig

Faktorisierung - Einleitung

- Die Sicherheit von RSA hängt maßgeblich davon ab, wie effizient eine Zahl faktorisiert werden kann
- Ausgangssituation: Eine Zahl n, von der bekannt ist, dass es keine Primzahl ist.
- Begriffsklärung:
  - Zerlegen einer Zahl n in zwei Faktoren a und b ungleich Eins (Faktorzerlegung)
    $$n = a \cdot b$$
  - Zerlegen einer Zahl n in ihre Primfaktoren (Primfaktorzerlegung)
    $$n = p_1^{e_1} \cdot p_2^{e_2} \cdot \ldots \cdot p_k^{e_k}$$
- Beispiel: $30 = 2 \cdot 3 \cdot 5$
- Eine Primfaktorzerlegung ist immer eindeutig
- Eine natürliche Zahl größer Eins ist entweder eine Primzahl oder zusammengesetzt

Faktorisierung - Verfahren

- Probedivision (Brute-Force)
- p-1-Methode von Pollard
- Pollard rho
- Fermat-Methode
- Basis: Quadratische Kongruenzen
  - Quadratisches Sieb
  - Dixons Random Squares
  - Zahlkörpersieb

Faktorisierung - Probedivision

- Naiver und einfacher Ansatz
- Annahme: n ist keine Primzahl

Faktorisierung - Probedivision

- Naiver und einfacher Ansatz
- Annahme: n ist keine Primzahl
- Prüfe für alle $2 \leq k \leq \lfloor \sqrt{n} \rfloor$, ob n teilbar durch k

Faktorisierung - Probedivision

- Naiver und einfacher Ansatz
- Annahme: n ist keine Primzahl
- Prüfe für alle $2 \leq k \leq \lfloor \sqrt{n} \rfloor$, ob n teilbar durch k
- Sinnvoll für Zahlen kleiner 10^{12}, für größere Zahlen zu langsam

Faktorisierung - Probedivision

- Naiver und einfacher Ansatz
- Annahme: n ist keine Primzahl
- Prüfe für alle $2 \leq k \leq \lfloor \sqrt{n} \rfloor$, ob n teilbar durch k
- Sinnvoll für Zahlen kleiner 10^{12}, für größere Zahlen zu langsam
- Modifikationen:
  - Es müssen nur ungerade Zahlen geprüft werden
  - Verwende eine Schranke B, um nur kleine Primfaktoren zu finden (typische (max.) Schranke ist $B = 10^6$)
  - Teste nur Primzahlen, die kleiner als $\lfloor \sqrt{n} \rfloor$ sind (Sieb des Eratosthenes)

Algorithmen im Detail - Pollard-(p-1)-Methode

- entwickelt von John Pollard
- spezieller Algorithmus
- geeignet für n mit einem B-glatten $p-1$
- Definition:
  Sei $B \in \mathbb{N}$. $F(B) = \{p \in \mathbb{P} : p \leq B\} \cup \{-1\}$
  Hat q nur Primfaktoren in $F(B)$, dann heißt q B-glatt.
  $F(B)$ heißt Faktorbasis.

Algorithmen im Detail - Pollard-(p-1)-Methode

- Grundlagen:
  Kleiner Satz von Fermat: $a^{p-1} \equiv 1 \ mod \ p$ für alle $a \in \mathbb{Z}$, falls p Primzahl.
  $\Rightarrow \quad a^k \equiv 1 \ mod \ p$, falls $p-1|k$
  Ist $a^k - 1$ nicht durch n teilbar, so ist $gcd(a^k - 1, n)$ ein echter Teiler von n
- nimm $k = B!$, wobei alle Primzahlpotenzen, die p-1 teilen, $\leq B$ sind

Algorithmen im Detail - Pollard-(p-1)-Methode

- Pseudo-Code:

$(p-1)Fact(n, B)\{$
$\quad a := 2$
$\quad for\ j\ from\ 2\ to\ B\ do\{$
$\quad\quad a := a^j\ mod\ n$
$\quad \}$
$\quad d := gcd(a-1, n)$
$\quad if\ \ \{1 < d < n\}\ \ then\ \ return(d)$
$\quad\quad else\ \ return(failure)$
$\}$

Algorithmen im Detail - Pollard-Rho-Methode

- entwickelt von John Pollard
- spezieller, probabilistischer Algorithmus
- effizient für kleinen Primfaktor
- Idee: generiere Folge von Pseudozufallszahlen, finde Kollision *mod p*
- berechne Folge $x_1, x_2, x_3, \ldots$ mittels

$$x_{i+1} = f(x_i) \ mod \ n, \quad i \geq 1$$

wobei

$$f(x) = x^2 + c, \quad c \neq 0, -2$$

x_1 bel. Startwert

Algorithmen im Detail - Pollard-Rho-Methode

- p sei kleinster Primfaktor von n
- p sei bekannt, dann gibt es $mod\ p$ nur p verschiedene Werte in der Folge, d.h. irgendwann wiederholt sich ein Wert (Kollision), die Folge wird periodisch

  $\Rightarrow \rho$
- Für $x, y \in \mathbb{Z}_n$ (aus der Folge) mit $x \neq y$ und $x \equiv y\ mod\ p$ gilt:

$$p \leq gcd(x - y, n) < n$$

  d.h gcd liefert nichttrivialen Teiler von n

Algorithmen im Detail - Pollard-Rho-Methode

- Beispiel: $n = 7171 = 71 \cdot 101$
  $f(x) = x^2 + 1, \quad x_1 = 1$

  x_i:
  1, 2, 5, 26, 677, 6557, 4105, 6347, 4903, 2218, 219, 4936, 4210, 4560, 4872, 375, 4377, 4389, 2016, 5471, 88, 574
  x_i *mod* 71:
  1, 2, 5, 26, 38, 25, 58, 28, 4, 17, 6, 37, 21, 16, 44, 20, 46, 58, 28, 4, 17, 6, ...
  Vorperiode der Länge 7, Periode der Länge 11

Algorithmen im Detail - Pollard-Rho-Methode

- p ist aber unbekannt:
  suche x_i und x_j aus der Folge mit $gcd(x_i - x_j, n) > 1$ und $x_i \neq x_j$ ($\Rightarrow \ gcd = n$)
  berechne $gcd(a - b, n), \quad a, b$ aus Folge
  wenn $gcd = 1$, nimm andere a,b
  wenn $1 < gcd < n$, echter Teiler gefunden
  wenn $gcd = n$, starte Verfahren neu mit neuem Startwert x_1
- nimm als a,b die Werte x_i, x_{2i} (spart viele Vergleiche)
- Laufzeit: $O(\sqrt{p})$ ($\Rightarrow$ Geburtstagsparadoxon)

Algorithmen im Detail - Pollard-Rho-Methode

- Pseudo-Code:

Pollard-Rho-Fact$(n, x_1, f)\{$

$x := x_1$

$y := f(x) \ mod \ n$

$p := gcd(x - y, n)$

$while \ \ \{p == 1\} \ \ do\{$

$x := f(x) \ mod \ n$

$y := f(y) \ mod \ n$

$y := f(y) \ mod \ n$

$p := gcd(x - y, n)$

$\}$

$if \ \ \{p == n\} \ \ then \ \ return($failure$)$

$else \ \ return(p)$

$\}$

Algorithmen im Detail - Pollard-Rho-Methode

- Beispiel: $n = 7171$
  $f(x) = x^2 + 1, \quad x_1 = 1$

i	x_i	x_{2i}	$gcd(x_i - x_{2i}, n)$		i	x_i
1	1	2	1		12	4936
2	2	26	1		13	4210
3	5	6557	1		14	4560
4	26	6347	1		15	4872
5	677	2218	1		16	375
6	6557	4936	1		17	4377
7	4105	4560	1		18	4389
8	6347	375	1		19	2016
9	4903	4389	1		20	5471
10	2218	5471	1		21	88
11	219	574	71		22	574

Algorithmen im Detail - Pollard-Rho-Methode

- Beispiel: $n = 7171 = 71 \cdot 101$
  $f(x) = x^2 + 1, \quad x_1 = 1$

  $x_i \ mod\ 71$:
  1 2 5 26 38 25 58 28 4 17 6 37 21 16 44 20 46 58 28 4 17 6 ...
  Vorperiode der Länge 7, Periode der Länge 11

- Faktorisierung von Fermatzahl $F_8 = 2^{2^8} + 1 = 2^{256} + 1$

Algorithmen im Detail - Fermat-Methode

- spezieller Algorithmus, nicht probabilistisch
- effizient, wenn die Differenz der Primfaktoren klein ist
- Voraussetzung: $n = p_1 \cdot p_2$
  mit p_1, p_2 ungerade und $p_1 < p_2$
- Dann ist: $p_2 - p_1 = 2 \cdot d$, $d \in \mathbb{N}$.
  Mit $x := p_1 + d$ und $y := d$ gilt folgendes:
  $n = (x - y)(x + y) = x^2 - y^2$ d.h. $x^2 - n = y^2 = d^2 > 0$,
  also $x > \sqrt{n}$ und $p_1 = x - d$, $p_2 = x + d$
- deshalb: probiere $x = [\sqrt{n}] + i, i = 1, 2, 3, ...$
  bis $x^2 - n$ eine Quadratzahl ist.

Algorithmen im Detail - Fermat-Methode

- Pseudo-Code:

Fermat-Fact(n){
$x := [\sqrt{n}]$
$d^2 := x^2 - n$
while $\{d^2$ *keine Quadratzahl*$\}$ *do*{
$x := x + 1$
$d^2 := x^2 - n$
}
$d := \sqrt{d}$
$p_2 := x + d$
$p_1 := x - d$
return(p_1, p_2)
}

Algorithmen im Detail - Fermat-Methode

- Beispiel:
  $n = 143$
  $\lfloor \sqrt{143} \rfloor + 1 = 12$ es gilt: $12^2 - 143 = 1$ ist ein Quadrat.
  $\Rightarrow \quad x = 12, \; y = 1$
  $\Rightarrow \quad 143 = (12 - 1)(12 + 1) = 11 \cdot 13$

Algorithmen im Detail - Quadratische Kongruenzen

Prinzip

Es sei für $x, y \in \mathbb{Z}$: $x^2 \equiv y^2 \ mod \ n$ und $x \not\equiv \pm y \ mod \ n$
Dann gilt:

$$n \mid x^2 - y^2 = (x - y)(x + y), \text{ aber } n \nmid x - y, \ n \nmid x + y$$

d.h. $gcd(x - y, n)$ und $gcd(x + y, n)$ sind echte Teiler von n

Problem: finde x, y mit $x^2 \equiv y^2 \ mod \ n$

Algorithmen im Detail - Quadratische Kongruenzen

Quadratisches Sieb:

- entwickelt von Carl Pomerance
- Laufzeit: $exp^{v(log\ \ n)^u(log\ \ log\ \ n)^{1-u}}$ mit $u = 1/2, v = 1 + o(1)$
- genereller Algorithmus
- probalistisch: mit WK $\geq 1/2$ liefert eine quadrat. Kongruenz einen nichttrivialen Faktor
- Es sei $f(x) = (x + m)^2 - n$ mit $m = \lfloor \sqrt{n} \rfloor$
  d.h. $(x + m)^2 \equiv f(x)\ mod\ n$
- finde s, so dass f(s) B-glatt ist, $B \in \mathbb{N}$ gewählt
- Multipliziere Kongruenzen so, dass auf der rechten Seite ein Quadrat steht
  links steht immer ein Quadrat

Algorithmen im Detail - Quadratische Kongruenzen

- Auswahl der Kongruenzen: welche Kongruenzen ergeben ein Quadrat?
  bei n groß: >100000 mögliche Kongruenzen
  stelle lineares Gleichungssystem/Kongruenzsystem über $\mathbb{Z}_2$ auf

Man hat:

$$f(x_1) = p_1^{e_{1,1}} \cdot \ldots \cdot p_w^{e_{1,w}}$$
$$f(x_2) = p_1^{e_{2,1}} \cdot \ldots \cdot p_w^{e_{2,w}}$$
$$\ldots$$
$$f(x_r) = p_1^{e_{r,1}} \cdot \ldots \cdot p_w^{e_{r,w}}$$

Algorithmen im Detail - Quadratische Kongruenzen

$$\Rightarrow \prod_{i=1}^{r} f(x_i)^{\lambda_i} = \prod_{j=1}^{w} p_j^{\sum_{k=1}^{r} \lambda_k e_{k,j}}, \quad \lambda_i = \begin{cases} 0, & \text{nimm Kongruenz i nicht;} \\ 1, & \text{nimm Kongruenz i.} \end{cases}$$

Damit das ein Quadrat ist, muss gelten:

$$\sum_{k=1}^{r} \lambda_k e_{k,j} \equiv 0 \ mod\ 2 \quad \text{für alle} \quad j \in \{1, ..., w\}$$

$\Rightarrow$GS mit w Gleichungen und r Unbekannten $\lambda_1, ..., \lambda_r$

- Finde mind. $|F(B)|$ Kongruenzen

Algorithmen im Detail - Quadratische Kongruenzen

- Wie findet man die Kongruenzen?
- 1. Möglichkeit: berechne f(s) für $s = 0, \pm1, \pm2, \pm3, ...$ und prüfe durch Probedivision auf B-Glattheit
  $\Rightarrow$ ähnlich Dixons Random Squares
- 2. Möglichkeit: fixiere Siebintervall $S = \{-C, ..., C\}$
  Ziel: suche alle $s \in S$, so dass $f(s)$ B-glatt ist
  berechne f(s) für alle $s \in S$
  $\forall p \in F(B)$:
  dividiere alle f(s) durch höchstmögliche p-Potenz
  f(s) ist B-glatt, wenn 1 oder -1 übrig bleibt

Algorithmen im Detail - Quadratische Kongruenzen

Optimierung: das Sieb

- Welche f(s) lassen sich durch p teilen?
- bestimme $s \in \{0, 1, ..., p-1\}$,so dass $p|f(s)$
- es gibt maximal 2 Lösungen für
  $f(s) = 0 \ mod \ p, \quad s \in \{0, 1, ..., p-1\}$
- wenn $p|f(s)$, dann $p|f(s+kp), \quad k \in \mathbb{Z}$
  $\Rightarrow$ vgl. Sieb des Eratosthenes
- man spart fast alle erfolglosen Probedivisionen

Algorithmen im Detail - Quadratische Kongruenzen

- Beispiel:

  $n = 7429 \qquad \Rightarrow m = 86, \quad f(x) = (x + 86)^2 - 7429$

  Es gilt:

$$
\begin{aligned}
f(-3) = 83^2 - 7429 = -540 &= -1 \cdot 2^2 \cdot 3^3 \cdot 5 &&\Rightarrow\ 83^2 \equiv -1 \cdot 2^2 \cdot 3^3 \cdot 5 \ mod\ 7429 \\
f(1) = 87^2 - 7429 = 140 &= 2^2 \cdot 5 \cdot 7 &&\Rightarrow\ 87^2 \equiv 2^2 \cdot 5 \cdot 7 \ mod\ 7429 \\
f(2) = 88^2 - 7429 = 315 &= 3^2 \cdot 5 \cdot 7 &&\Rightarrow\ 88^2 \equiv 3^2 \cdot 5 \cdot 7 \ mod\ 7429
\end{aligned}
$$

$$\Rightarrow \quad (87 \cdot 88)^2 \equiv (2 \cdot 3 \cdot 5 \cdot 7)^2 \ mod\ 7429$$

Setze $\quad x = 87 \cdot 88 \ mod\ 7429 = 227$

und $\quad y = 2 \cdot 3 \cdot 5 \cdot 7 \ mod\ 7429 = 210$

$$\Rightarrow \quad gcd(x - y, 7429) = gcd(17, 7429) = 17 \text{ ist echter Teiler von n}$$

Fazit - Faktorisierung

- es gibt spezielle und generelle Algorithmen
- es gibt deterministische und probabilistische Algorithmen
- es gibt noch keinen Algorithmus mit polynomieller Laufzeit (außer Shor-Algorithmus)

Fazit - Faktorisierung

Stand der RSA Faktorisierungsrekorde

Zahl	Bits	Preis in $	faktorisiert	Gruppe um	GIPS-J.	Algorithmus
RSA-100	332	-	April 1991	A. K. Lenstra	0, 007	Quadr. Sieb
RSA-110	365	-	April 1992	A. K. Lenstra	0, 075	Quadr. Sieb
RSA-120	399	-	Juni 1993	T. Denny	0, 830	Quadr. Sieb
RSA-129	429	100	April 1994	A. K. Lenstra	5	Quadr. Sieb
RSA-130	432	-	April 1996	A. K. Lenstra	1	Zahlkörpersieb
RSA-140	465	-	Feb. 1999	H. te Riele	2	Zahlkörpersieb
RSA-155	512	-	Aug. 1999	H. te Riele	8, 4	Zahlkörpersieb
RSA-160	532	-	April 2003	J. Franke		Zahlkörpersieb
RSA-576	576	10.000	Dez. 2003	J. Franke	13, 2	Zahlkörpersieb
RSA-640	640	20.000	Nov. 2005	J. Franke	170	Zahlkörpersieb
RSA-703	703	30.000	offen			
...			...			
RSA-1024	1024	100.000	offen			

Quelle: *http : //www.informatik.tu − darmstadt.de/KP/lehre/ws*0506*/vl/skript/factor.pdf* ,S. 90

Quellen (1)

- Douglas R. Stinson: Cryptography Theory and Practice Second Edition. Chapman & Hall/CRC 2002
- Johannes Buchmann: Einführung in die Kryptographie. Springer Verlag 2003
- Klaus Denecke: Algebra und diskrete Mathematik für Informatiker. Teubner 2003 (mehr zum kleinen Fermatschen Satz aus der Restklassensicht)
- Gilbert Brands: Verschlüsselungsalgorithmen. Vieweg 2002 (anspruchsvoll mit vielen Informationen und mathematischen Details, nähert sich über Gruppentheorie)
- Wikipedia: `http://de.wikipedia.org/wiki/Primzahlen`
- Mathematik.de: `http://www.mathematik.de/mde/information/landkarte/zahlen/primzahlen.html`

Quellen (2)

- Arndt Brünner: Rechner für große (und kleine) Zahlen mit hoher Genauigkeit unter `http://www.arndt-bruenner.de/mathe/java/rechnergz.htm`
- Axel Wagner: Mathematische Grundlagen, Square-and-Multiply-Alg. und andere Java-Applets unter `http://www.saar.de/~awa/kry02.htm`
- Nuria Brede: Primzahltests. Vortrag im Rahmen des Kryptographie-Seminars im Sommersemester 2005
- Suche nach den größten Primzahlen der Welt: `http://www.mersenne.org/prime.htm`
- 25. Algorithmus der Woche: Das Sieb des Eratosthenes: `http://www-i1.informatik.rwth-aachen.de/~algorithmus/Algorithmen/algo25/algo25.pdf`

Quellen (3)

Philipp Dörsek: Primzahlen in der angewandten Zahlentheorie

Johann Wiesenbauer: Primzahltests und Faktorisierungsalgorithmen I. Internationale Mathematische Nachrichten Nr. 186. April 2001.

Annegret Weng: Kryptographie. Skript zur Vorlesung am Fachbereich Mathematik der Johannes Gutenberg Universität Mainz. Juli 2004.

The Prime Pages. http://primes.utm.edu/index.html

Wikipedia: Lucas-Lehmer test for Mersenne numbers. http://en.wikipedia.org/wiki/Lucas%E2%80%93Lehmer_test_for_Mersenne_numbers

Wikipedia: Lucas-Lehmer test. http://en.wikipedia.org/wiki/Lucas-Lehmer_test

Michael E. Pohst: Der Lucas-Lehmer Test. http://www.math.tu-berlin.de/~kant/Mersenne/MersenneVortrag.pdf

Daniel J. Bernstein: Proving Primality After Agrawal-Kayal-Saxena. http://cr.yp.to/papers/aks.ps

Wikipedia: AKS-Primzahltest. http://de.wikipedia.org/wiki/AKS-Methode

Volker Kaatz: Faktorisierung. Vortrag im Rahmen des Kryptographie-Seminars im Sommersemester 2005

Quellen (4)

- Arjen K. Lenstra: Integer Factoring
  http : *//modular.fas.harvard.edu/edu/Fall*2001/124/*misc/*
  arjen_lenstra_factoring.pdf
- Bianca Schröder: Ein Faktorisierungsalgorithmus
  http : *//krypt.cs.uni − sb.de/teaching/seminars/ss*1998/
  writeups/schroeder.ps.gz
  (Quadratisches Sieb)
- Katja Schmidt-Samoa: Das Number Field Sieve - Entwicklung, Varianten und Erfolge
  http : *//www.informatik.tu − darmstadt.de/KP/*
  theses/Katja_Schmidt − Samoa.diplom.pdf
- http://www.informatik.tu-darmstadt.de/KP/lehre/ws0506/vl/skript/factor.pdf