Fachhochschule Wedel
Studiengang: Wirtschaftsinformatik
Sommersemester 2016

Seminararbeit

Seminar IT-Sicherheit

Efficient Number-Theoretic Algorithms

vorgelegt von: Antonio Galeazzi
Matrikelnummer: 100183

eingereicht bei: Prof. Dr. Gerd Beuster

Abgabetermin: 01.06.2016

Inhaltsverzeichnis

1. Einleitung

Die vorliegende Arbeit setzt sich mit dem Kapitel 1.3 „Efficient Number-Theoretic Algorithms“ aus dem Buch „Cryptanalytic Attacks on RSA“ von Song Y. Yan auseinander. [1]
Dieser Abschnitt behandelt eine Reihe von Algorithmen, die in der Zahlentheorie Verwendung finden, und darüber hinaus unter anderem auch in der Kryptographie eine herausragende Rolle spielen.
Zu den angesprochenen Algorithmen zählen unter anderem der bekannte euklidische Algorithmus in seiner erweiterten Form („Euclid's algorithm“), Kettenbrüche („continued fraction algorithm“), schnelles modulares Potenzieren („fast modular exponentiations“), der Chinesische Restsatz („Chinese Remainder Theorem“), Gruppenoperationen auf elliptischen Kurven („group operations on elliptic curves“) sowie Primzahltests („primality testing“).
Nachfolgend werden die Primzahltests den Schwerpunkt bilden. Dabei soll zum einen auf das eigentliche Primzahlproblem („primality testing problem“) und zum anderen auf mögliche Lösungsansätze und Problematiken eingegangen werden.
Verdeutlicht wird dies insbesondere anhand zweier Algorithmen: dem Miller-Rabin-Test und dem Fermatschen Primzahltest.
Diese werden nacheinander vorgestellt und anschließend hergeleitet bzw. erklärt, um dann abschließend das Vorgehen anhand eines Beispiels zu verdeutlichen.
Zuletzt wird auf mögliche Implementationen der Tests eingegangen, um die Leistungsfähigkeit und Funktionsweise zu veranschaulichen.

2. Primzahltest – Überblick

2.1. Das Primzahltestproblem

Primzahlen spielen in der Zahlentheorie und insbesondere in der Kryptographie eine herausragende Rolle. So werden beispielsweise für das RSA-Verfahren, das zum Verschlüsseln (und Entschlüsseln) von Nachrichten genutzt wird, zwei voneinander unterscheidbare Primzahlen p und q benötigt, um zunächst das sogenannte RSA-Modulus N zu bilden.

$$N = p \cdot q \, mit \; p \, , \; q \in Primzahlen$$

Die Primzahlen, die dabei für gewöhnlich zum Einsatz kommen, haben eine Größenordnung von etwa 1000 Stellen in binärer Darstellung, was einer Größenordnung von näherungsweise 10^{300} entspricht.
Es existiert kein Algorithmus, der es ermöglicht, Primzahlen zu generieren, sondern lediglich Verfahren, die eine Zahl bestimmen, die mit einer großen Wahrscheinlichkeit die Eigenschaften einer Primzahl aufweisen.
Daher muss eine entsprechend große Zahl generiert werden, ohne die Gewissheit zu haben, dass es sich dabei tatsächlich um eine Primzahl handelt. Um dies sicherzustellen, wird anschließend ein (oder mehrere) Primzahltest durchgeführt.
Formal kann das Primzahltestproblem folgendermaßen definiert werden:

$$Eingabe: N \in \mathbb{Z}_{>1}$$
$$Ausgabe: wahr \, , wenn \, N \, eine \, Primzahl \, ist \, ; \; falsch \, , andernfalls$$

Für eine gegebene natürliche Zahl N ist zu entscheiden, ob es sich bei dieser um eine Primzahl handelt oder nicht. [1]

2.2. Überblick über bekannte und praxisrelevante Verfahren

2.2.1. Bekannte Verfahren

Es gibt viele verschiedene Ansätze, um Primzahltests zu realisieren. Zu den bekanntesten zählen sicherlich die folgenden:

- Die *Probedivision* stellt den einfachsten Primzahltest dar. Dabei wird für eine Zahl N getestet, ob diese durch eine Primzahl p mit $2 \leq p \leq \sqrt{n}$ zu teilen ist. Sollte N durch eine dieser Zahlen teilbar sein, so kann N offensichtlich keine Primzahl sein. Andernfalls handelt es sich bei N um eine Primzahl.
  Dieser Verfahren ist für entsprechend große Zahlen (wie sie für das RSA-Verfahren beispielsweise benötigt werden) selbstverständlich nicht praktikabel.

- Auch das *Sieb des Eratosthenes* stellt einen simplen Test dar, durch den eine Liste von Primzahlen erzeugt wird. Dies geschieht, indem aus einer Liste von Zahlen (bei 1 beginnend) bis zu einer zu definierenden Grenze alle Vielfachen einer Primzahl entfernt werden. Die verbleibenden sind dann Primzahlen. Auch dieses Verfahren ist in der Praxis für entsprechend große Zahlen nicht anwendbar. [3][6]

2.2.2. Praxisrelevante Verfahren

Zu den Primzahltests, die in der Praxis eine breite Anwendung finden gehören unter anderem der Miller-Rabin-Test und der fermatsche Primzahltest. [4]
Primzahltests können grundsätzlich in probabilistische und deterministische Tests unterteilt werden. Während deterministische Primzahltests eine genaue Aussage darüber treffen können, ob es sich bei einer Eingabe N um eine Primzahl handelt oder nicht, kann dies bei probabilistischen Primzahltests nur mit einer gewissen Wahrscheinlichkeit festgestellt werden.
Die präzisere Aussage eines deterministischen Tests hat allerdings eine höhere Laufzeit zur Folge, so dass die Durchführung eines solchen Tests in der Regel mehr Zeit beansprucht, als die eines probabilistischen Tests.
Der Miller-Rabin-Test repräsentiert einen probabilistischen Primzahltest, wohingegen der AKS-Test zu den deterministischen zählt.
In der Praxis wird häufig zunächst ein probabilistischer Primzahltest durchgeführt, um festzustellen, ob es sich bei N grundsätzlich um eine Primzahl handeln könnte, um dann durch darauf aufbauende weitere deterministische Verfahren die Gewissheit zu erhalten. [4]
Eine weitere Möglichkeit besteht darin, einen probabilistischen Test, wie den Miller-Rabin-Test, mit verschiedenen Parametern durchzuführen, wodurch dessen Fehlerwahrscheinlichkeit mit jedem weiteren Durchgang erheblich gesenkt werden kann. [1]

3. Fermatscher Primzahltest

3.1. Vorgehen

Der fermatsche Primzahltest basiert im Wesentlichen auf dem kleinen fermatschen Satz, der folgendes besagt:

$$a^{p-1} \equiv 1 \pmod{p}$$

Diese Kongruenz wird von jeder Primzahl *p* und jeder dazu teilerfremden natürlichen Zahl *a* erfüllt, wobei aus dem alleinigen Erfüllen der Kongruenz nicht geschlossen werden kann, dass es sich tatsächlich um eine Primzahl handelt, da diese auch von einigen anderen Zahlen erfüllt werden kann.
Um den fermatschen Primzahltest durchzuführen, wird eine zu testende natürliche Zahl *N* als Eingabe benötigt, für die dann im folgenden entschieden wird, ob sie zusammengesetzt oder möglicherweise eine Primzahl ist.
Es ist dann eine Basis *a* mit $1 < a < N$ zu wählen. Sollten *a* und *N* nicht teilerfremd sein, so folgt umgehend, dass *N* keine Primzahl sein kann. Sollten *a* und *N* teilerfremd sein, so ist zu überprüfen, ob die obenstehende Kongruenz erfüllt werden kann. Sollte sie nicht erfüllt werden können, so gilt, dass *N* zusammengesetzt ist, wohingegen ein Nichterfüllen lediglich zur Folge hat, dass *N* eine Primzahl sein könnte.
Zahlen, die den fermatschen Primzahltest mit einer Basis *a* bestehen, bezeichnet man auch als fermatsche Pseudoprimzahlen zur Basis *a*.
Grundsätzlich gilt, dass je häufiger der Test mit unterschiedlichen Basen *a* durchgeführt und von der eingegebenen Zahl *N* bestanden wird, die Wahrscheinlichkeit, eine zusammengesetzte Zahl als solche zu erkennen, deutlich steigt.
Hervorzuheben sind in diesem Zusammenhang die sogenannten Carmichael-Zahlen. Diese zeichnen sich durch die besondere Eigenschaft aus, den fermatschen Primzahltest für alle teilerfremden Basen *a* zu bestehen, obwohl sie keine Primzahlen sind.
Diese werden bei der Vorstellung des fermatschen Tests und Miller-Rabin Tests noch genauer betrachtet werden.

3.2 Beispiel

Betrachtet man beispielsweise die Primzahl 5 gilt:

$$2^4 \equiv 3^4 \equiv 4^4 \equiv 1 \pmod{5}$$

5 besteht demnach den Test für alle möglichen Basen *a*.
Betrachtet man hingegen die natürliche Zahl 8 mit der teilerfremden Basis 5 gilt:

$$5^7 \equiv 5 \pmod{8}$$

Demnach handelt es sich bei 8 um eine zusammengesetzte Zahl.

4. Kleiner fermatscher Satz

4.1. Beweis

Gemäß des kleinen fermatschen Satzes gilt die nachfolgende Kongruenz für sämtliche Primzahlen *p* und natürliche Zahlen *a*:

$$a^{p-1} \equiv 1 \ (mod \ \ p)$$

bzw.

$$a^{p} \equiv a \ (mod \ \ p)$$

Die Gültigkeit der Kongruenz lässt sich folgendermaßen beweisen:

Sei *W* die Menge der natürlichen Zahlen, die kleiner als *p* sind. Demnach gilt:

$$W = \{1, 2, 3, \ ..., \ p-1\}$$

Man multipliziere nun jedes Element der Menge mit $a \ (mod \ \ p)$, um die Menge *X* zu erhalten:

$$X = \{a \ mod \ \ p, \ 2a \ mod \ \ p, \ 3a \ mod \ \ p, \ ..., \ (p-1)a \ mod \ \ p\}$$

Da *a* nicht von *p* geteilt wird, kann gefolgert werden, dass alle in *X* enthaltenen Elemente ungleich null sind.
Darüber hinaus folgt aus der Tatsache, dass *a* und *p* teilerfremd sind, dass alle Elemente in *X* voneinander unterscheidbar sind, und es somit keine zwei gleichen Elemente gibt.

Fasst man die vorangehenden Eigenschaften zusammen, lässt sich festhalten, dass alle *p-1* Elemente in *X* positiv und voneinander unterscheidbar sind.

Man kann nun folgern, dass die Menge *X* aus den Elementen der Menge *W* besteht, wobei die Reihenfolge nicht bekannt, aber auch nicht von Bedeutung ist.
Entscheidend ist jedoch, dass das Produkt über alle Elemente der Menge *X* gleich dem Produkt über alle Elemente der Menge *W* modulo p ist.

Es gilt demnach:

$$a \cdot 2a \cdot 3a \cdot ... \cdot (p-1)a \equiv [1 \cdot 2 \cdot 3 \cdot ... \cdot (p-1)] \ (mod \ \ p)$$

Der Ausdruck lässt sich zusammenfassen zu:

$$a^{p-1}(p-1)! \equiv (p-1)! \ (mod \ \ p)$$

Eliminiert man auf beiden Seiten der Kongruenz die Fakultät folgt:

$$a^{p-1} \equiv 1 \ (mod \ \ p)$$

Dies ist die Kongruenz, die zu zeigen war. [2]

5. Miller-Rabin-Test

5.1. Überblick

Wie auch beim bereits vorgestellten fermatschen Primzahltest handelt es sich beim Miller-Rabin-Test nicht um einen deterministischen, sondern um einen probabilistischen Primzahltest.
Es kann also nicht mit Sicherheit entschieden werden, ob es sich bei einer natürlichen Zahl N um eine Primzahl handelt.
Der Algorithmus kann feststellen, dass eine Zahl definitiv *keine* Primzahl ist, aber lediglich behaupten, dass eine Zahl *möglicherweise* eine Primzahl ist. Es kann demnach passieren, dass Zahlen, die keine Primzahl sind, fälschlicherweise für eine „Primzahl" gehalten werden. Es ist jedoch nicht möglich, dass eine Primzahl den Test nicht besteht. Sollte eine Zahl den Test also nicht bestehen, kann mit Gewissheit gesagt werden, dass es sich bei der Eingabe um keine Primzahl handelte.
Der Vorteil gegenüber deterministischen Primzahltests ist die deutlich höhere Geschwindigkeit des Miller-Rabin-Tests. Mehrmaliges Ausführen des Tests sorgt zusätzlich dafür, dass die Wahrscheinlichkeit, eine Eingabe fälschlicherweise für eine Primzahl zu halten, verschwindend gering wird. [5]

5.2. Algorithmus

Der Miller-Rabin-Test läuft grundsätzlich nach dem folgenden Schema ab:

1. Sei N eine ungerade Zahl und die Basis b aus dem Intervall $1 < b < N$.
   Wähle j und d, so dass $N - 1 = 2^j d$ gilt, wobei d eine ungerade Zahl ist.

2. Setze $i = 0$ und $y = b^d \pmod N$.

3. Wenn $i = 0$ und $y = 1$ oder $y = N - 1$, dann beende den Algorithmus und gib „N ist wahrscheinlich eine Primzahl" aus.
   Wenn $i < j$, setze $y = 1$ und fahre mit 5. fort.

4. Setze $i = i + 1$. Wenn $i < j$, setze $y = y^2 \pmod N$ und fahre mit 3. fort.

5. Beende den Algorithmus und gib „N ist keine Primzahl" aus. [1]

5.3. Erklärung

Der Miller-Rabin-Test basiert im Wesentlichen auf dem kleinen fermatschen Satz, der folgendes besagt:

$$a^p \equiv a \,(mod\ \ p),$$

wobei *a* eine ganze Zahl und *p* eine Primzahl ist.

Unter der Annahme, dass *a* kein Vielfaches von *p* ist, kann man obige Kongruenz auch schreiben als:

$$a^{p-1} \equiv 1 \,(mod\ \ p)$$

Aus diesem Satz lässt sich der Satz über starke Pseudoprimzahlen herleiten.
Gemäß des obigen Satzes gilt für jede Primzahl:

$$p \mid a^{p-1} - 1$$

Demnach ist $a^{p-1}-1$ durch *p* teilbar.

Durch eine umgekehrte Anwendung der 3. binomischen Formel erhält man:

$$a^{p-1} - 1 = \left(a^{\frac{p-1}{2}} + 1\right)\left(a^{\frac{p-1}{2}} - 1\right)$$

p muss also entweder $a^{\frac{p-1}{2}} + 1$ oder $a^{\frac{p-1}{2}} - 1$ teilen.

Sollte $\frac{p-1}{2}$ eine gerade Zahl sein, so kann eine weitere Aufspaltung vorgenommen werden, die zu folgendem Ergebnis führt:

$$a^{p-1} - 1 = \left(a^{\frac{p-1}{2}} + 1\right)\left(a^{\frac{p-1}{4}} + 1\right)\left(a^{\frac{p-1}{4}} - 1\right)$$

Setzt man $p - 1 = 2^j d$ so lassen sich diese Aufspaltungen genau *j*-mal wiederholen und es stellt sich heraus, dass *p* einen der *s + 1* Faktoren teilen muss.

Damit ergibt sich nun der Satz über starke Pseudoprimzahlen:

$$a^d \equiv 1 \,(mod\ \ p)$$

oder

$$a^{d \cdot 2^r} \equiv -1 \,(mod\ \ p)$$

für ein $r\ mit\ 0 \le r < s\ und\ \ p - 1 = 2^j d.$

Erfüllt eine Zahl diese Kongruenz, so wird sie als starke Pseudoprimzahl zur Basis *a* bezeichnet. Mit dem Miller-Rabin-Test lassen sich nun ebendiese Pseudoprimzahlen bestimmen, in dem das Verfahren sich den Satz über starke Pseudoprimzahlen zunutze macht.
Ausgehend von a^d wird überprüft, ob $a^d \equiv 1 \pmod{p}$ *oder* $a^d \equiv -1 \pmod{p}$ gilt.
Anschließend findet höchstens *j-1*-maliges Quadrieren von a^d statt, und es wird in jedem Durchgang überprüft, ob $a^{d \cdot 2^r} \equiv -1 \pmod{p}$ gilt. Sobald man auf *-1* trifft, kann daraus geschlossen werden, dass alle nachfolgenden Quadrate *1* als Rest bei Teilung durch *p* aufweisen.
Die Wahrscheinlichkeit, dass eine Zahl *p* eine starke Pseudoprimzahl zur Basis *a* ist, aber keine Primzahl ist, liegt im schlechtesten Fall bei etwa $\frac{1}{4}$. Durch *k*-maliges Ausführen des Miller-Rabin-Tests mit jeweils unterschiedlichen Basen *a* kann die Fehlerwahrscheinlichkeit auf $\frac{1}{4^k}$ verringert werden. [2]

5.4. Beispiel

Betrachtet wird im folgenden die erste Carmichael-Zahl 561. Es gilt:

$$561 = 3 \cdot 11 \cdot 17$$

Es handelt sich bei 561 also offensichtlich um keine Primzahl. Allerdings besteht 561 den fermatschen Primzahltest für alle teilerfremden Basen.

Es gilt:

$$561 - 1 = 2^4 \cdot 35$$

Damit folgt: $j = 1$ *und* $d = 35$. Die verwendete Basis a sei $a = 2$.

Der Term $2^{560} - 1$ lässt sich folgendermaßen zerlegen:

$$2^{560} - 1 = (2^{280}+1)(2^{140}+1)(2^{70}+1)(2^{35}+1)(2^{35}-1)$$

Es ist nun im ersten Schritt zu prüfen, ob $a^d = 2^{35} \equiv \pm 1 \pmod{561}$ gilt.
Dies ist nicht der Fall, da tatsächlich $2^{35} \equiv 263 \pmod{561}$ gilt.
Quadriert man nun den Term 2^{35} erhält man $(2^{35})^2 = 2^{70}$. Es ist nun zu prüfen, ob $2^{70} \equiv -1 \pmod{561}$ gilt, was aufgrund von $2^{70} \equiv 166 \pmod{561}$ nicht der Fall ist.
Es werden nun noch zwei weitere Quadrierungen durchgeführt, die beim ersten Mal 2^{140} und beim zweiten Mal 2^{280} liefern.
Es gilt $2^{140} \equiv 67 \pmod{561}$ und $2^{280} \equiv 1 \pmod{p}$, womit folgt, dass 561 den Miller-Rabin-Test nicht bestanden hat und dem zur Folge definitiv keine Primzahl sein kann.

6. Implementationen

6.1. Fermatscher Primzahltest

Der fermatsche Primzahltest lässt sich exemplarisch in Python folgendermaßen implementieren:

```
1   def testfermat(n,a):
2       return a**(n-1) % n != 1
3
4   def fermat(n,k):
5       t = range(2,n-1)
6       for i in range(k):
7           a = t[ZZ.random_element(len(t))]
8           if testfermat(n,a):
9               return True
10          t.remove(a)
11      return False
```

6.2. Miller-Rabin-Test

Eine mögliche Python-Implementation des Miller-Rabin-Tests ist die Folgende:

```
1   def testrabinmiller(n,a):
2       e = 0
3       q = n-1
4       while q % 2 == 0:
5           e += 1
6           q = q/2
7       if a**q % n == 1:
8           return False
9       for i in range(e):
10          if a**(q*2**i) % n == n-1:
11              return False
12      return True
13
14  def rabinmiller(n,k):
15      t = range(2, n-1)
16      for i in range(k):
17          a = t[ZZ.random_element(len(t))]
18          if testrabinmiller(n,a):
19              return True
20          t.remove(a)
21      return False
```

6.3. Auswertung

Möchte man nun die Genauigkeit der Primzahltests für sämtliche Zahlen zwischen 10 und 9999 testen, so lässt sich das durch die folgenden Aufrufe der zuvor genannten Skripte bewerkstelligen, durch den alle Zahlen ausgegeben werden, die Tests bestanden haben, ohne jedoch tatsächlich eine Primzahl zu sein.

Fermatscher Primzahltest:

```
1   for i in range(10,10000):
2       if not fermat(i,2) and not is_prime(i):
3           print(i)
```

Der Aufruf liefert als Ergebnis die folgenden Zahlen: 65, 91, 703, 1729, 1921, 2821, 5611 und 8911.

Miller-Rabin-Test:

```
1   for i in range(10,10000):
2       if not rabinmiller(i,2) and not is_prime(i):
3           print(i)
```

Dieser Aufruf liefert als Ergebnis die folgenden Zahlen: 3367 und 8321.

Beide Tests wurden mit zwei unterschiedlichen Basen durchgeführt. Es zeigt sich, dass der Fermat-Test von acht Zahlen, die keine Primzahlen sind, bestanden worden ist, wohingegen der Miller-Rabin-Test nur von zwei Pseudoprimzahlen bestanden wurde.
Berücksichtigt man hierbei, dass lediglich zwei Basen zur Durchführung der Tests verwendet worden sind, und selbst dadurch schon ein brauchbares Ergebnis zustande kam, lässt sich leicht vorstellen, dass noch häufigeres Ausführen mit unterschiedlichen Basen die Fehlerwahrscheinlichkeit der Tests, insbesondere des Miller-Rabin-Tests, verschwindend gering werden lässt.

8. Literaturverzeichnis

Bücher

[1] Song Y. Yan: Cryptanalytic Attacks on RSA, Springer Science + Business Media LLC, 2008

[2] William Stallings: Cryptography and Network Security, 5.Auflage, Prentice Hall, 2011

Internetquellen

[3] http://cr.yp.to/primetests.html (Stand 04.10.2016)

[4] http://mathworld.wolfram.com/PrimalityTest.html (Stand 04.10.2016)

[5] http://mathworld.wolfram.com/Rabin-MillerStrongPseudoprimeTest.html (Stand 04.10.2016)

[6] http://mathworld.wolfram.com/StrongPseudoprime.html (Stand 04.10.2016)

[7] http://tcs.rwth-aachen.de/lehre/PRICS/WS2006/gamper.pdf (Stand 04.10.2016)

[8] https://www.cs.cornell.edu/courses/cs4820/2010sp/handouts/MillerRabin.pdf (Stand 04.10.2016)