Teil 5 Schwer invertierbare Funktionen, RSA-Algorithmus

Bei allen bisher besprochenen *symmetrischen* Chiffriersystemen benutzten Alice und Bob einen öffentlichen Chiffrieralgorithmus und einen geheimen Schlüssel. Über den Schlüssel mußten die beiden sich vorher verständigen und mit ihm wurde verschlüsselt und entschlüsselt. – Dies war das jahrhundertealte Grundprinzip der Kryptografie.
Bei unsymmetrischen Kryptosystemen sind die Mechanismen grundsätzlich anders. Neben dem Algorithmus ist auch der Schlüssel öffentlich!
Jeder Teilnehmer des Systems besitzt einen individuellen, öffentlichen Schlüssel.
Will Alice eine vertrauliche Botschaft an Bob schicken, so sucht sie sich in einem öffentlichen Verzeichnis (sozusagen *im Telefonbuch*) Bobs Schlüssel und verschlüsselt damit ihre Botschaft und schickt sie an Bob.
Bob kann die Botschaft entschlüsseln, während der Angreifer dazu nicht in der Lage ist. Es nützt dem Angreifer auch nichts, sich in den Besitz von Bobs Schlüssel zu bringen, der ist ohnehin öffentlich bekannt.
Und die vollständige, hinter dem System stehende Mathematik ist auch jedermann zugänglich, ist nicht sonderlich kompliziert - und wird im folgenden erläutert.

Diese fast unwahrscheinlich klingenden Public-Key-Chiffriersysteme wurden in den siebziger Jahren des vergangenen Jahrhunderts entdeckt. Sie resultieren aus Erkenntnissen von Theoretischer Informatik/Komplexitätstheorie und benutzen Eigenschaften schwer invertierbarer Funktionen.
Wobei es sich bei diesen *schwer invertierbaren Funktionen* handelt, wollen wir zunächst erläutern.

Die gängigen mathematischen Funktionen sind leicht invertierbar:

Beispiel
Es ist leicht, eine Zahl zu quadrieren. Es ist auch nicht sehr schwer, die Wurzel zu ziehen, d.h. von einer Zahl y festzustellen, von welcher Zahl x sie das Quadrat ist. Wir benutzen ein Halbierungsverfahren:

Sei y = 49 : x=32 : x^2 zu groß
x=16 : x^2 zu groß
x=8 : x^2 zu groß
x=4 : x^2 zu klein
x=6 : x^2 zu klein
x=7 : x^2=49, stimmt

In der Praxis wird man die Wurzel effizienter bestimmen, aber schon dieses Halbierungsverfahren geht schnell.

Ebenso wie die Quadratfunktion sind andere Funktionen wie Potenzfunktion, Exponentialfunktion etc. leicht invertierbar.

Daß man es andererseits mit *schwer* invertierbaren Operationen täglich zu tun hat, zeigt dieses Beipiel:

Beispiel (schwer invertierbare Funktion)
Es ist leicht, im Telefonbuch die Telefonnummer einer Person zu finden (weil die Namen alphabetisch geordnet sind).
Es ist aber schwer, im gleichen Telefonbuch die zu einer bestimmten Nummer gehörende Person zu identifizieren. Hierfür ist keine andere Lösung erkennbar, als das ganze Telefonbuch (im Mittel das halbe Buch) durchzusehen.

Wir führen einige weitere Beispiele für schwer invertierbare Operationen oder Funktionen an:

Beispiel (Rucksackproblem)
Es ist leicht, aus einer Päckchenmenge einige bestimmte Päckchen herauszunehmen und diese gemeinsam auszuwiegen.
Bei der umgekehrten Fragestellung wären zu einem vorgegebenen Gesamtgewicht geeignete Päckchen in der Päckchenmenge zu finden, die zusammen genau dieses Gewicht haben. Nehmen wir an, wir haben Päckchen mit den Gewichten

3 , 9 , 15 , 25 , 30 , 63

Ein Paket mit Gesamtgewicht 67 kann man daraus zusammenstellen, wie man durch Ausprobieren feststellt:

67 = 3+9+25+30

Tatsache ist: Vom Gesamtgewicht auf die Päckchen zu schließen, ist eine *schwere* Aufgabe, die für große Datensätze selbst mit Rechnereinsatz derzeit nicht zu schaffen ist!
Das Letztere ist deswegen so überraschend, weil das Rucksackproblem so einfach und für jedermann sofort verständlich formuliert werden kann. - Der Nachweis der großen Komplexität dieses Rucksackproblem war eines der ersten Ergebnisse der *Komplexitätstheorie.*

Beispiel (Faktorisierung)
Es ist leicht (geht bei gängigen Rechnern auf Tastendruck), zwei 300-stellige Primzahlen miteinander zu multiplizieren.
Es ist jedoch *schwer* (dauert derzeit *Jahre* auf den größten Rechnern), aus dem Produkt die beiden Primzahlen wieder zu rekonstruieren.

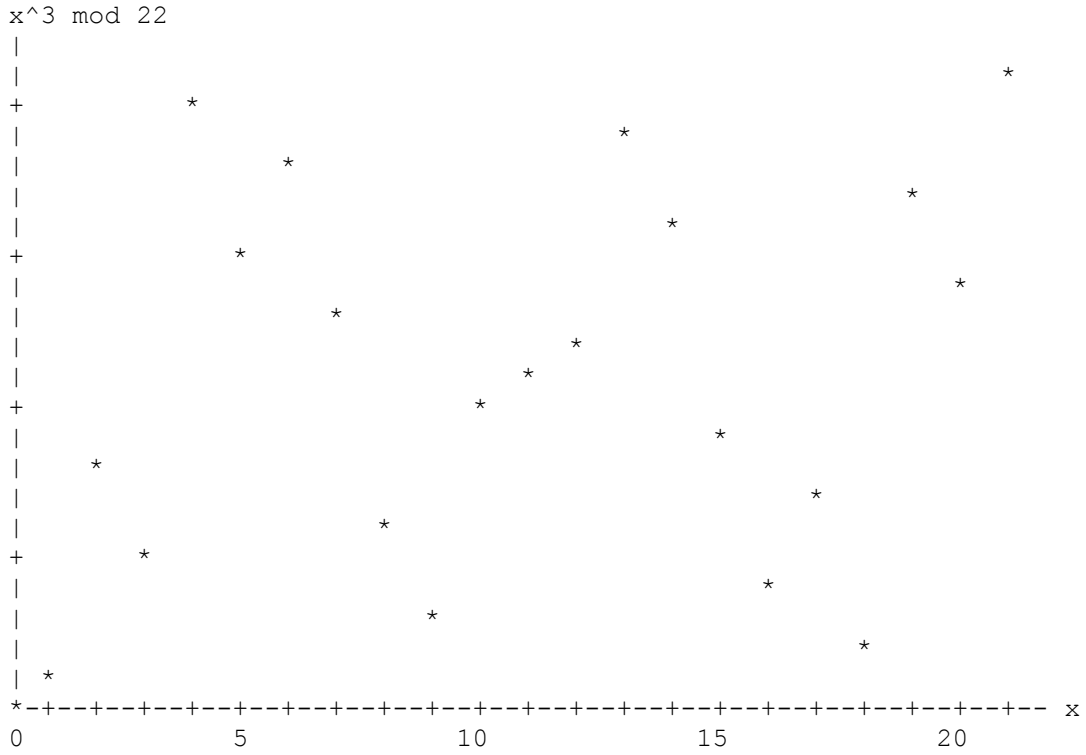
Beispiel (diskrete Wurzel)
Die oben als leicht invertierbar erkannten Potenz- und Exponentialfunktionen werden schwer invertierbar, sobald man *modulo n* rechnet, wobei n eine feste natürliche Zahl ist. Wir demonstrieren das für die Potenzfunktion.

Als Modul wählen wir n=22 und als Exponent e=3.

```
m   : 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21
m^3 : 0  1  8  5 20 15 18 13  6  3 10 11 12 19 16  9  4  7  2 17 14 21
```

Beispielsweise ist 5^3 = 125 = 5*22 + 15 ≡ 15 mod 22

Man erkennt zunächst: Potenzierung mit e=3 liefert eine umkehrbare Abbildung für die Restklassen modulo 22.
(Bei Potenzierung mit 4 ist das durchaus nicht so, man vergleiche die Tabelle in den Ergänzungen.)



Es ist offensichtlich leicht, eine Zahl modular mit 3 zu potenzieren (es erfordert zwei Multiplikationen und Reduktionen modulo 22).
Es ist hingegen schwer, die dritte modulare Wurzel zu ziehen. Die oben beschriebene Halbierungsmethode funktioniert nicht. Der Grund ist, daß die dritte modulare Potenz nicht monoton ist, sondern ein völlig unregelmäßiges Verhalten hat (siehe Skizze).

Um diese dritte modulare Wurzel von y zu finden, ist keine andere Methode erkennbar, als die Liste der dritten modularen Potenzen durchzugehen, bis man y antrifft:

```
read y
x:=1
while  x^3/=y mod n and x<n
do
   x := x+1
end
return x
```

Diese Methode funktioniert natürlich, sie läuft aber auf Ausprobieren aller Möglichkeiten hinaus und ist daher überhaupt nicht effizient. Der Rechenaufwand der Schleife ist O(n), effizient wäre logarithmischer Aufwand O(log n).

Randbemerkungen
(i) Die modulare Inverse man mit einer ähnlichen (ineffizienten) Schleife bestimmt werden. Für die modulare Inverse gibt es jedoch mit dem Euklidischen Algorithmus eine effiziente Alternative.
(ii) Damit die modulare Potenz schwer umkehrbar wird, ist es nicht einmal nötig, den Exponenten besonders groß zu wählen. Schon das modulare Quadrat ist schwer umkehrbar. Hierauf gründet sich die Sicherheit des *Fiat-Shamir-Protokolls* (siehe Teil 6).
(iii) Ebenfalls schwer berechenbar ist der diskrete Logarithmus, also die Umkehrung der modularen Exponentialfunktion,. Auf diese Tatsache gründet sich das *Diffie-Hellmann-Protokoll* (siehe Skript Kryptografie 2).

Schwer invertierbare Funktionen bilden den Kern aller derzeit eingesetzten Public-Key Kryptosysteme.
Das erste veröffentlichte Public-Key-System war das Diffie-Hellmann Protokoll. Es dient nicht zur Verschlüsselung, sondern zum Schlüsselaustausch in hybriden Systemen wie *PGP (Pretty Good Privacy)* (siehe Teil 6).
Ein weiteres, zunächst vielversprechendes Public-Key-System gründete sich auf die Komplexität des Rucksackproblems. Diese *Knapsack-Chiffre* (siehe Artikel von Diffie, S. 563/64) wurde in der Zwischenzeit jedoch gebrochen.

Das erste praktisch eingesetzte und derzeit immer noch genutzte Public-Key Kryptosystem war die *RSA-Chiffre* (benannt nach den Autoren: Ronald Rivest, Adi Shamir und Leonard Adleman). Dieses System gründet sich auf die praktische Unumkehrbarkeit von modularer Potenz und Primzahlmultiplikation.
Den RSA-Algorithmus wollen wir im Folgenden beschreiben.

Die beim praktischen Einsatz des RSA-Algorithmus' benutzten Primzahlen haben mindestens 300 Dezimalstellen. Die als Modul benutzten Zahlen sind das Produkt zweier solcher Primzahlen, haben also Bitlängen von 1024 oder größer. Zur Funktionsweise des RSA-Algorithmus zunächst ein winziges Beispiel:

Beispiel (für RSA-Algorithmus)
Der Klartext besteht aus einer Sequenz natürlicher Zahlen, die alle kleiner sind als der vorgewählte Modul. Für das Beispiel wählen wir

Bobs öffentliche Schlüssel, bestehend aus:
Modul n = 22
Exponent e = 3 (e: encrypt)

Der Klartext, den Alice an Bob schicken will, besteht aus Zahlen, die alle kleiner als n sind:

p = 3,7,2,17,3,4 (p: plaintext)

Zum Verschlüsseln sucht Alice im *Telefonbuch* Bobs Schlüssel und potenziert jede Zahl ihres Klartextes mit dem Exponenten e=3 und reduziert modulo n (also modulo 22).
Der resultierende Geheimtext, den sie an Bob schickt, ist:

c = 5,13,8,7,5,20 (c: Ciphertext)

Man kann sich leicht vorstellen, daß es für den Angreifer aussichtslos ist, den Geheimtext zu entschlüsseln, wenn der Modul 1024 Bit lang ist. Allerdings denkt man: Die gleiche Schwierigkeit hat doch auch Bob.
Der kennt jedoch ein Geheimnis seines Moduls n. Er kennt die Zerlegung des Moduls in die beiden Primzahlen
n = p*q = 2*11
Mit dieser Kenntnis b errechnet Bob zunächst (bzw. hat das schon lang vorher gemacht) die Größe:

$\Phi(n) = (p-1)*(q-1) = 1*10 = 10$

$\Phi(n)$ ist die sogenannte *Eulersche Funktion* (siehe unten).
Für Bob ist es kein Problem, mit Hilfe des euklidischen Algorithmus die modulare Inverse d vom Exponenten e bezüglich $\Phi(n)$ zu bestimmen:

$e*d \equiv 1 \bmod \Phi(n)$ ($\Phi(n)=10$)

Im Beispiel erhält Bob

d = 7 (d: Decrypt)

Kontrolle: $e*d = 3*7 = 21 \equiv 1 \bmod 10$ (ok)

Diese Zahl d ist nur Bob bekannt und ist sein Schlüssel zum Entschlüsseln der Geheimbotschaft c.
Bob potenziert nun (im Prinzip genau wie Alice) jede Zahl vom Geheimtext c mit einem Exponenten, allerdings nicht wie Alice mit e, sondern mit seinem geheimen d=7. Bob erhält (man vergl. die Tabelle in den Ergänzungen):

3,7,2,17,3,4 = p

Man vergleicht mit p und muß zugeben, daß dies in der Tat der Klartext von Alice ist.

Man beachte: Man sieht leicht ein, daß Bobs Exponent e nicht völlig beliebig sein darf. Denn natürlich ist es nötig, daß Bobs Verschlüsselungsfunktion

x |---> x^e mod n

umkehrbar ist. Das ist beispielsweise nicht der Fall bei Modul n=5 und Exponent e=2. Das Quadrat mod 5 ist nicht umkehrbar, da es nicht injektiv ist:

$2^2 \equiv 4 \bmod 5$ und auch $3^2 \equiv 4 \bmod 5$

[Für den RSA-Algorithmus wäre Exponent e=2 nicht möglich, denn die Zahl 2 ist nicht teilerfremd zum (geradzahligen) $\Phi(n) = (p-1)*(q-1)$.]

Wir beschreiben die RSA-Chiffre nun allgemeinen:

RSA-CHIFFRIERUNG
Bob wählt zwei verschiedene Primzahlen p und q und berechnet
n = p*q (n: mindestens 1024 Bit lang)
Er berechnet dann
$\Phi(n) = (p-1)*(q-1)$
Er bestimmt eine zu $\Phi(n)$ teilerfremde Zahl e und ferner (mit dem euklidischen Algorithmus) deren Inverse d mod $\Phi(n)$.
Bob veröffentlicht n und e.
Alice will eine aus Zahlen bestehende Klartextnachricht p an Bob senden. Die Zahlen sind alle kleiner als n.
Alice sucht sich Bobs Schlüssel n und e aus dem öffentlichen Verzeichnis.
Sie potenziert jede der Zahlen des Klartextes p mit e modulo n und erhält so ihre Geheimnachricht c, die sie an Bob schickt.
Bob potenziert jede der Zahlen der empfangenen Geheimbotschaft c mit d modulo n und erhält damit Alices Klartext p zurück.

Randbemerkung: Wenn umgekehrt Bob eine Nachricht an Alice schicken will, dann bedient sich Bob Alices öffentlichen Schlüssels, d.h. die beiden vertauschen einfach ihre Rollen.

Beispiel (Originalbeispiel Diffie)
Bob
wählt die Primzahlen p = 17 , q = 31 .
Dann errechnet er: n = 17*31 = 527
$\Phi(n) = 16*30 = 480$
Als Schlüssel e
kann er wählen: e = 7 (ist teilerfremd zu $\Phi(n)$)

Als Inverse d zu e mod $\Phi(n)$ erhält Bob d=343.

Kontrolle: $7*343 = 2401 = 5*480+1 \equiv 1 \bmod 480$

Bob gibt n und e öffentlich bekannt.

Alice
hat eine Klartextnachricht p=2
Sie berechnet c = 2^e = 2^7 = 128 und schickt c=128 an Bob

Bob
entschlüsselt. Er berechnet 128^343 mod 527 :

$$
\begin{array}{lllllll}
128^{343} & 256 & 64 & 16 & 4 & 2 & 1 \\
= 128 & * 128 & * 128 & * 128 & * 128 & * 128 & \bmod 527 \\
= 35 & * 256 & * 35 & * 101 & * 47 & * 128 & \bmod 527 \\
= 2 & \bmod 527 & & & & &
\end{array}
$$

Also Bob hat den Klartext richtig rekonstruiert (man beachte: mit 8+5 = 13 Multiplikationen).

Geht man die Einzelheiten des Verfahrens durch, so erkennt man: Verschlüsseln durch Alice und Entschlüsseln für Bob ist prinzipiell leicht, und dies auch noch bei Moduln, die aus 1024 oder mehr Bits bestehen.

Ist das Verfahren aber auch sicher? Der Angreifer kennt die (öffentlichen) Schlüssel n und e. Es fragt sich, ob er mit diesen Informationen etwas anfangen kann?
Er könnte versuchen vorzugehen wie Bob. Als erstes hätte er den Modul n in die beiden Primzahlen n=p*q zu zerlegen, und genau daran scheitert er: Die Faktorisierung einer aus zwei Primzahlen zusammengesetzten Zahl n in die beiden Faktoren p und q ist ein anerkannt schwieriges Problem. Nach dem derzeitigen Wissensstand der Komplexitätstheorie ist es für den Angreifer ein aussichtsloses Unterfangen, von n auf p und q zu schließen, wenn n mehr 1024 Bit oder länger ist.

Der Angreifer könnte auch direkt versuchen, die Verschlüsselungsfunktion

x |---> x^e mod n

umzukehren, d.h. die e-te modulare Wurzel zu ziehen. Aber bei der modularen Potenz handelt es sich um eine schwer invertierbare Funktion, die (nach derzeitigem Kenntnisstand) praktisch unumkehrbar ist.

Zu den exotisch anmutenden Einzelheiten des RSA-Algorithmus werden im Folgenden nähere mathematische Erläuterungen gegeben. Zuvor aber dieses Zitat aus [Beutelspacher]: "Daß man ernsthaft daran denkt, ein Kryptosystem einzuführen, dessen Stärke darauf beruht, daß kein Mathematiker es bisher geschafft hat, einen vernünftigen Faktorisierungsalgorithmus anzugeben, ist eine massive Beleidigung der Mathematik."

Um das Funktionieren des RSA-Algorithmus einzusehen, ist es erforderlich, sich mit einigen Grundtatsachen aus Zahlentheorie und (mathematischer) Gruppentheorie vertraut zu machen.

Man erinnere sich (Teil 1), daß durch die Kongruenzrelation *mod n* die Menge der ganzen Zahlen $\mathbb{Z}$ in das Restklassensystem $\mathbb{Z}_n$ eingeteilt wird. Von diesen Restklassen interessieren uns einige besonders:

Definition (prime Restklasse)
Gegeben sei eine natürliche Zahl n. Die Zahlen, die eine multiplikative Inverse mod n haben, sind nach den Ergebnissen von Teil 2 gerade die Zahlen, die zu n teilerfremd sind.
Die zugehörigen Restklassen sind die *primen Restklassen* mod n, wir bezeichnen sie mit $\mathbb{Z}_{n'}$. Die Anzahl der Elemente von $\mathbb{Z}_{n'}$ ist die *Eulersche Funktion* $\Phi(n)$.

Für n=22 erhält man

$\mathbb{Z}_{n'} = \{1, 3, 5, 7, 9, 13, 15, 17, 19, 21\}$ (n=22)

Inverse: 1 15 9 19 5 17 3 13 7 21

Die modulare Inverse von 9 ist 5, denn $9*5 = 45 \equiv 1 \bmod 22$.
Für n=22 hat $\mathbb{Z}_{n'}$ 10 Elemente, also $\Phi(22)=10$.
Das Restklassensystem $\mathbb{Z}_{n'}$ zusammen mit der modularen Multiplikation hat einige wichtige Eigenschaften:

(G1) Abgeschlossenheit: Das Produkt x*y (=y*x) zweier Elemente von $\mathbb{Z}_n$, ist wieder Element von $\mathbb{Z}_n$,.

(G2) Assoziativitätsgesetz: Für $x,y,z \in \mathbb{Z}_n$, gilt
$$(x*y)*z = x*(y*z)$$

(G3) Existenz des neutralen Elements: Für alle Elemente x aus $\mathbb{Z}_n$, gilt $1*x = x$

(G4) Existenz der Inversen: Für jedes Element $x \in \mathbb{Z}_n$, gibt es das inverse Element inv(x) mit
$$x*inv(x) = 1 .$$

(K) Kommutativität: Auf die Reihenfolge der Faktoren kommt es nicht an: $x*y = y*x$ für $x,y \in \mathbb{Z}_n$,

Die Regeln G2,G3,G4 und K verstehen sich von selbst. Die Gültigkeit von G1 illustrieren wir an einem Beispiel:
Die Zahl 3 hat eine Inverse, nämlich 15,
die Zahl 19 hat eine Inverse, nämlich 7
Dann hat auch 3*19 eine Inverse, nämlich 7*15 . Wir kontrollieren:

$$(3*19)*(7*15) = 3*(19*7)*15 \equiv 3*15 \equiv 1 \bmod 22$$

Was hier für das Restklassensystem $\mathbb{Z}_n$, nachgewiesen wurde, sind die *Gruppenaxiome*: Wann immer in einer Menge bezüglich einer Operation die obigen Regeln gelten, spricht man von einer *(kommutativen) Gruppe.*

Satz (prime Restklassengruppe)
Für jedes n bilden die primen Restklassen $\mathbb{Z}_n$, bezüglich der modularen Multiplikation eine *(kommutative) Gruppe,* d.h. es gelten die Regeln G1 bis G4 und K.
Die Anzahl der Elemente von $\mathbb{Z}_n$, ist die *eulersche Funktion* $\Phi(n)$ von n.

In der Gruppentheorie ist nun die folgende Tatsache bekannt:

Satz
Sei (G,*) eine kommutative Gruppe mit g Elementen (mit *Ordnung* g). Dann sind alle g-ten Potenzen gleich 1:
$$x^g = 1 \quad \text{für jedes Element } x \in G$$

Beweis: Sei $x_1, x_2, x_3, \ldots$ (*)
eine komplette Auflistung der Gruppenelemente.
Wir wählen ein x (eines der x_i) fest und multiplizieren alle Elemente der Liste mit diesem x:
$$x*x_1,\ x*x_2,\ x*x_3, \ldots \qquad (**)$$

Wir behaupten: Die Produkte in (**) sind dann auch alle verschieden.
Wäre nämlich etwa $x*x_1 = x*x_2$, so könnten wir von links mit inv(x) malnehmen:

$$\text{inv}(x)*x*x_1 = \text{inv}(x)*x*x_2, \text{ also } x_1=x_2 ,$$

was nicht stimmt.
Also stehen in (*) und in (**) dieselben Elemente. Wenn wir die alle miteinander malnehmen, so kommt auch das gleiche heraus:

$$\Pi\ x_i \ = \ \Pi\ x*x_i$$

Daraus folgt $\Pi\ x_i \ = \ x^g * \Pi\ x_i$

Multiplikation von rechts mit sämtlichen Inversen liefert dann

$$1 = x^g$$ //

Der letzte Satz angewandt auf die Gruppe der primen Restklassen $\mathbb{Z}_n'$ liefert:

Eulerscher Satz Sind die natürlichen Zahlen x und n teilerfremd, so gilt:

$$x^{\Phi(n)} \equiv 1 \mod n \quad \text{falls ggt}(x,n)=1$$

Man beachte: Aus $x^{\Phi(n)} \equiv 1 \mod n$ folgt

$$x^{\Phi(n)+1} \equiv x \mod n.$$

Hier wird schon deutlich, daß man durch Potenzieren einer Zahl x mit geeignetem Exponenten die Ausgangszahl reproduzieren kann, und das war ja der Angelpunkt des RSA-Algorithmus.

Wir brauchen später die Eulersche Funktion für zwei Spezialfälle:

Satz
(i) Für eine Primzahl p gilt:

$$\Phi(p) = p-1$$

(ii) Für ein Produkt n=p*q von unterschiedlichen Primzahlen gilt

$$\Phi(n) = (p-1)*(q-1)$$

Beweis
Zu (i): Alle Zahlen 1,2, ... ,p-1 sind teilerfremd zu p.

Zu (ii): *Nicht* teilerfremd zu n=p*q sind

p , 2p, ... , (q-1)p , q*p und
q , 2q, ... , (p-1)q

also insgesamt q + p-1 Zahlen. Die übrigen Zahlen sind zu n teilerfremd, und das sind n-q-p+1 = (p-1)*(q-1) Stück. //

Folgerung (Spezialfall des Satzes von Euler)
Ist p eine Primzahl, so gilt

$$x^{p-1} \equiv 1 \bmod p \quad \text{für alle } x=1,2,\ldots,p-1$$

Das letzte Ergebnis gibt nähere Einsicht beim Satz von Euler. Ist dort n=p*q das Produkt von zwei Primzahlen, so gilt die Aussage des Satzes zwar nicht für alle p*q möglichen x, aber doch für die allermeisten, nämlich für (p-1)*(q-1) Stück.

In den Ergänzungen findet man für n=22 die Werte x^Φ(n) = x^10 aufgelistet [Φ(22) = Φ(2*11) = 1*10 = 10].

Nun sind fast alle Einzelheiten für den RSA-Algorithmus versammelt:

Alice verschlüsselt durch Potenzieren mit e.
Bob entschlüsselt durch Potenzieren mit d.

Insgesamt wird also mit e*d potenziert und es soll wieder der Klartext herauskommen:

$$x^{e*d} \equiv x \bmod n$$

Für teilerfremde x und n darf man wegen des Satzes von Euler im Exponenten mod Φ(n) rechnen:

$$x^{\alpha*\Phi(n)+1} \equiv x \bmod n \quad \text{für} \quad \alpha \in \mathbb{N}$$

Durch Vergleich wird die folgende Wahl für die Exponenten e und d nahegelegt:

$$e*d = \alpha*\Phi(n)+1$$

Modulo Φ(n) gelesen ist das

$$e*d \equiv 1 \bmod \Phi(n)$$

Konsequenz: Man sollte e und d so wählen, daß sie mod Φ(n) zueinander invers sind. Macht man das, dann resultiert die gewünschte Dechiffrierung:

$$x^{e*d} = x^{\alpha*\Phi(n)+1} \equiv x \bmod n$$

Diese Argumentation ist *beinahe* komplett, sie gilt für teilerfremde x und n. - Was noch fehlt, ist der folgende Hilfssatz, der die spezielle Form der Zahl n=p*q als Produkt zweier Primzahlen ausnutzt:

Hilfssatz
Sei n das Produkt zweier verschiedener Primzahlen p und q. Dann gilt

$$x^{\alpha*\Phi(n)+1} \equiv x \bmod n \quad \text{für } \textit{alle } x \quad (\alpha \in \mathbb{N})$$

Beweis Wir haben einige Fälle zu unterscheiden.
(i) x ist teilerfremd zu n.
Dann ist der Satz von Euler anwendbar. Die linke Seite der Gleichung reduziert sich zu 1*x, also stimmt die Gleichung.
(ii) Sei x nicht teilerfremd zu n. Dann könnte sein
(iia) x ist Vielfaches von n. Dann gilt die Gleichung, weil beide Seiten nämlich kongruent zu 0 sind.
(iib) Ist x nicht Vielfaches von n=p*q aber auch zu n nicht teilerfremd, so ist
(*) x teilerfremd zu der einen Primzahl (etwa p) aber
(**) x Vielfaches der anderen Primzahl (dann q).
Nun wird die explizite Formel für $\Phi(n)$ benutzt, und es wird zunächst mod p (und nicht mod n) weitergerechnet:

$$x^{\alpha*\Phi(n)+1} = (x^{\Phi(n)})^{\alpha} * x = (x^{p-1})^{\alpha(q-1)} * x = 1^{\alpha(q-1)} * x \equiv x \bmod p$$

Die vorletzte Gleichung der letzten Kette folgt aus dem Satz von Euler mit p an Stelle von n unter Beachtung von (*) und der Formel $\Phi(p)=p-1$.

Die Gleichung des Satzes gilt also modulo p.
Wegen (**) gilt sie sowieso mod q (mod q sind beide Seiten 0).
Dann muß (warum?) die Gleichung aber auch modulo p*q, d.h. mod n gelten. //

Die Mathematik zur RSA-Chiffre ist damit weitgehend komplett, und es ist nötig, einiges zum praktischen Einsatz des RSA-Systems zu sagen.
Hat Bob die Primzahlen p und q, so benötigt er die zu $\Phi(n)$ teilerfremde Zahl e. Die zu bestimmen ist kein Problem, jede Primzahl größer als $\Phi(n)$ ist geeignet.
Ferner benötigt Bob die Inverse d von e modulo $\Phi(n)$. Die kann Bob effizient mit dem euklidischen Algorithmus ausrechnen.

Alice und Bob haben den Klartext bzw. den empfangenen Geheimtext mit e bzw. d modulo n zu potenzieren. Das ist wie oben am Beispiel vorgeführt numerisch effizient machbar.

Bleibt die Frage, wie Bob an seine Primzahlen kommt. Auch die zu bestimmen ist kein Problem. Es gibt Algorithmen, die effizient und mit *großer Zuverlässigkeit* (wenn auch nicht mit absoluter Sicherheit) von einer vorgegebenen Zahl entscheiden, ob sie prim ist. Diese Algorithmen gehen aus von dem schon erwähnten Spezialfall der Eulerschen Formel:

$$\textbf{p prim} \Rightarrow x^{p-1} \equiv 1 \bmod p \quad \text{für alle } x=1,2,\ldots,p-1$$

Es wäre praktisch, wenn in der umgekehrten Richtung folgendes gälte

(*) **n nicht prim** $\Rightarrow x^{n-1} \equiv 1 \bmod n$ nur für *wenige* x

Das stimmt aber leider nicht. Es gibt sogenannte *Carmichael-Zahlen* n, die zerlegbar sind und bei denen dennoch

$x^{n-1} \equiv 1 \bmod n$ für die allermeisten x.

Die kleinste dieser Carmichael-Zahlen ist 561. Man kann nachrechnen:

$$x^{560} \equiv 1 \bmod 561 \quad \text{für alle zu 561 teilerfremden x}$$

und trotzdem ist 561 nicht prim. 561 ist nämlich durch 3 teilbar.

Es ist nun möglich, die Probleme mit den Carmichael-Zahlen zu überwinden, indem man oben bei (*) schärfer analysiert. Auf dieser Basis wurden Primzahltests entwickelt, die schnell sind und mit *hoher Zuverlässigkeit* funktionieren.
[Siehe Ergänzungen Teil 6 und Bruce Schneier: Applied Cryptography, 2. Auflage, 1996]

Gegen die Sicherheit des RSA-Systems gibt es einen naheliegenden Einwand: Die Sicherheit des Systems hängt daran, daß es im Durchschnitt astronomisch lange Rechenzeiten erfordert, ein Primzahlprodukt in die beiden Faktoren zerlegen. Der Angreifer könnte einwenden: “Aber wenn ich Glück habe, dann schaffe ich es relativ früh oder gar auf Anhieb.“
Dieser Einwand ist zunächst nicht falsch, daher ist in im Buch von Beutelspacher diese Glückswahrscheinlichkeit genauer quantifiziert. Der Autor kommt zum Ergebnis: Ein Primzahlprodukt *mit Glück* zu faktorisieren ist *viel* unwahrscheinlicher, als *dreimal nacheinander* 6 Richtige im Lotto zu haben.

Wenn man also durch Faktorisieren eines Primzahlschlüssels nur einige Millionen Gewinn erhofft, so wäre es *viel* sinnvoller, den dafür nötigen Einsatz beim Zahlenlotto zu riskieren.

Ergänzungen zur Kryptografie / 5

1) Beim Standardbeispiel im Text war der Modul n=22=2*11.
Dann ist p=2, q=11 und $\Phi(n) = 1*10 = 10$
Ferner hatten wir e=3 und d=7 gewählt.
Im Text wurde auf die Einträge der folgenden Tabelle Bezug genommen:

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

2) Was ist der größte gemeinsame Teiler von 2166 und 6099 ?

3) Man bestimme ganze Zahlen x und y mit $1 = 17*x + 55*y$

4) Wie viele Multiplikationen reichen, um m^{21} auszurechnen?

5) Man faktorisiere die Zahl x=14.803, wenn man weiß, daß
   (i) x das Produkt von zwei Primzahlen ist und daß
   (ii) $\Phi(x)=14.560$ gilt.

6) In der Vorlesung wurde behauptet, daß man aus der Telefonnummer und anhand des Telefonbuches nur sehr schwer den zugehörigen Namen herausbekommen kann.
Es gibt aber eine ganz einfache Methode, den Namen zu finden.
Welche?