

Primzahlen und ihr Nutzen für die Kryptographie

PHILIPP MORITZ

$$f_k(m) = m^a \pmod{n}$$

$$a = \prod_{k=1}^n p_k^{e_k}$$

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

$$a^b = \prod_{k=0}^n (a^{2^k})^{b_k}$$

$$m^{ed} \equiv m \pmod{n}$$

F a c h a r b e i t

aus dem Fach Mathematik

Primzahlen und ihr Nutzen für die Kryptographie

Verfasser: Philipp C. Moritz

Leistungskurs: Mathematik 3 M

Kursleiter: Herr Werner Seitz

Bearbeitungszeitraum: 12/II bis 13/I

Abgabetermin: 30. Januar 2009

Ergebnisse:

Erzielte Punkte schriftlich:
(einfache Wertung)

Erzielte Punkte mündlich:
(einfache Wertung)

.....

(Unterschrift des Kursleiters)

Inhaltsverzeichnis

1	Die Bedeutung von Primzahlen in der Kryptographie	1
2	Einführung in die Zahlentheorie	2
2.1	Die Teilbarkeitsrelation und ihre Eigenschaften	2
2.2	Division mit Resten	3
2.3	Der größte gemeinsame Teiler und Teilerfremdheit	3
2.4	Primzahlen und Primfaktorzerlegung	5
3	Kongruenzen und Restklassen	6
3.1	Definition und Eigenschaften von Kongruenzen und Restklassen	6
3.2	Rechnen mit Kongruenzen	7
3.3	Verändern des Moduls	7
4	Zahlentheoretische Algorithmen	8
4.1	Der (erweiterte) Algorithmus von Euklid	8
4.2	Die modulare Exponentiation	10
4.3	Das Lösen modularer Gleichungen	12
4.3.1	Das Finden von multiplikativen Inversen	12
4.3.2	Der chinesische Restsatz	12
5	Verschlüsselung mit dem RSA-Algorithmus	13
5.1	Der kleine Satz von Fermat	13
5.2	Berechnung der Schlüssel und Verschlüsselung	13
6	Primzahltests und Primzahlerzeugung	14
6.1	Probedivision und Sieb des Eratosthenes	14
6.2	Der Fermat-Test	15
6.3	Der Miller-Rabin-Test	17
6.3.1	Eine verschärfte Version des Fermat-Tests	17
6.3.2	Mehrere Zeugen sind noch zuverlässiger: Der Miller-Rabin-Test	20
7	Die Sicherheit der RSA-Verschlüsselung	20
A	Implementierung der Algorithmen in Common Lisp	22
A.1	Zahlentheoretische Algorithmen	22
A.2	Der RSA-Algorithmus	22
A.3	Die Suche nach Primzahlen	23
A.4	Praktische Anwendung des Verfahrens	25
B	Komplexität der Algorithmen	25
C	Nachweis der übernommenen Beweise	27

1 Die Bedeutung von Primzahlen in der Kryptographie

Bei der Verschlüsselung wird eine Botschaft in *Klartext* mithilfe eines Verschlüsselungsverfahrens, das sich eines *Schlüssels* bedient, in einen schwer zu interpretierenden *Geheimtext* umgewandelt. Man unterscheidet grundsätzlich zwei verschiedene Arten der Verschlüsselung, die *symmetrische* und die *asymmetrische*.

Im Falle eines symmetrischen Kryptosystems verwendet man den gleichen Schlüssel zur Ver- und Entschlüsselung, was allerdings den Nachteil in sich birgt, dass der Schlüssel auf einem sicheren Weg vom Sender, traditionell als „Alice“ bezeichnet, zum Empfänger „Bob“ übertragen werden muss. Ein unbefugter Dritter, evil „Eve“, könnte den Schlüssel bei der Übertragung abfangen und hätte damit ungehinderten Zugang zur Kommunikation von Alice und Bob (siehe Abb. 1).

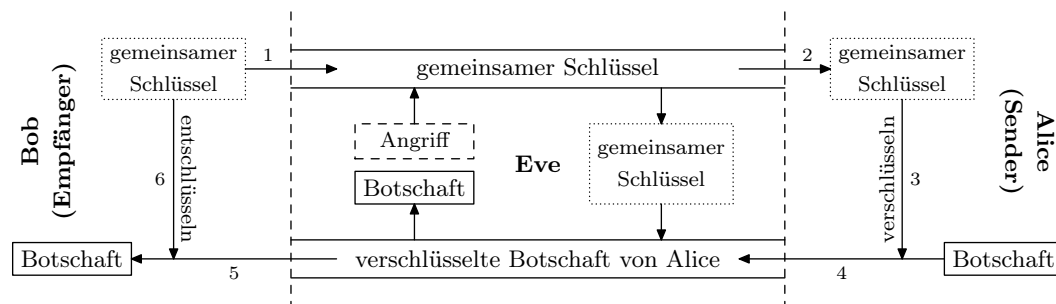


Abb. 1: Schema der symmetrischen Verschlüsselung und Angriffsfall

Einen Ausweg aus dieser Situation bietet die asymmetrische Verschlüsselung, bei der mit einem Schlüsselpaar gearbeitet wird, das aus zwei Teilen, einem geheimen (dem *privaten Schlüssel*) und einem nicht geheimen (dem *öffentlichen Schlüssel*), besteht. Der private Schlüssel darf auf keinen Fall in Eves Hände gelangen, der öffentliche Schlüssel wird zu jedermanns Verfügung freigegeben. Alice verschlüsselt ihre Botschaft mit dem öffentlichen Schlüssel von Bob und überträgt sie an den Empfänger. Der Geheimtext kann *nur* von Bob mit dessen privatem Schlüssel gelesen werden (siehe Abb. 2).

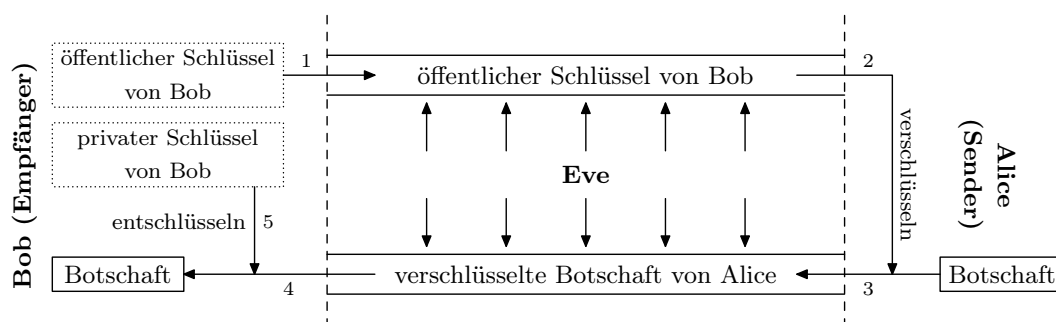


Abb. 2: Schema der asymmetrischen Verschlüsselung

Nun drängt sich natürlich sofort die Frage auf, wie man ein solches Schlüsselpaar konstruieren kann. Um eine Antwort zu geben, muss zuerst das Konzept der *Einwegfunktion* eingeführt werden, auf dem die asymmetrische Verschlüsselung basiert.

Definition 1. Eine *Einwegfunktion* f besitzt die Eigenschaft, dass $f(x)$ „einfach“, der Wert der Umkehrfunktion $f^{-1}(x)$ dagegen „schwierig“ zu berechnen ist.

Diese Definition ist nicht in einem streng mathematischen Sinn zu verstehen; es müsste erst geklärt werden, was „einfach“ und „schwierig“ überhaupt bedeutet, was den Rahmen dieser Arbeit sprengen würde. In der Tat ist die Existenz solcher Einwegfunktionen sogar ein ungelöstes Problem¹. Eine intuitive Vorstellung über Einwegfunktionen gewinnt man durch Betrachtung der Abbildung $f : \text{Name} \rightarrow \text{Telefonnummer}$ in einem Telefonbuch. Ein $f(x)$ ist schnell nachgeschlagen, für $f^{-1}(x)$ muss dagegen das gesamte Buch durchsucht werden. Hat man aber ein zusätzliches Hilfsmittel, in diesem Falle ein Telefonbuch, das nach Nummern sortiert ist, so kann $f^{-1}(x)$ problemlos bestimmt werden. Somit ist es möglich, f als öffentlichen und f^{-1} als privaten Schlüssel herzunehmen.

Die Funktionsweise des RSA-Verfahrens², das wir betrachten wollen, basiert darauf, dass es wesentlich einfacher ist, zwei große Primzahlen miteinander zu multiplizieren, als eine große Zahl in ihre Primfaktoren zu zerlegen. Eine unserer Grundannahmen lautet folglich „Eine hinreichend große Zahl ist in vertretbarer Zeit nicht faktorisierbar.“

Wie man aus dieser Idee eine Einwegfunktion basteln kann, ist der Inhalt dieser Arbeit. Wir werden zuerst in Abschnitt 2 einige grundlegende Sätze der Zahlentheorie kennen lernen. Abschnitt 3 führt in die Kongruenzrechnung ein. Aufbauend auf diese Grundlagen werden in Abschnitt 4 einige wichtige zahlentheoretische Algorithmen behandelt, um dann in Abschnitt 5 das RSA-Verfahren zu erläutern. In Abschnitt 6 betrachten wir Möglichkeiten, Primzahlen zu erzeugen, die zur Konstruktion der Schlüssel nötig sind, bevor wir in Abschnitt 7 abschließend auf die Sicherheit des RSA-Verfahrens eingehen.

2 Einführung in die Zahlentheorie

2.1 Die Teilbarkeitsrelation und ihre Eigenschaften

Ein zentraler Begriff der Zahlentheorie ist die „Teilbarkeit“ zweier ganzer Zahlen a und b . Anschaulich ist zumindest für natürliche Zahlen m und n klar, dass m genau dann durch n teilbar ist, wenn die Division m/n keinen Rest lässt. Auf ganze Zahlen verallgemeinert:

Definition 2. Eine ganze Zahl a ist genau dann ein *Teiler* der ganzen Zahl b , wenn eine ganze Zahl q existiert, sodass $b = a \cdot q$. Man sagt dann „ a teilt b “ oder „ b ist ein Vielfaches von a “ und schreibt $a \mid b$. Ist a kein Teiler von b , so notiert man $a \nmid b$.

Aus dieser Definition folgt unmittelbar, dass jede ganze Zahl a die 0 teilt, weil $0 = a \cdot 0$. Die Zahl 0 ist jedoch nur Teiler von sich selbst, da aus $a = 0 \cdot q$ sofort $a = 0$ folgt. Wir wollen nun einige einfache Regeln beweisen.

¹aus der Existenz von Einwegfunktionen folgt $P \neq NP$, die Frage $P \stackrel{?}{=} NP$ ist aber bekanntlich ungeklärt

²benannt nach den Erfindern des Verfahrens, RON RIVEST, ADI SHAMIR und LEN ADLEMAN

Satz 1 (Grundlegende Eigenschaften der Teilbarkeitsrelation). *Für alle ganzen Zahlen a, b, c, k und h gelten folgende Gesetze der Teilbarkeitsrelation.*

$$a \mid b \quad \text{und} \quad a \mid c \quad \implies \quad a \mid kb + hc, \quad (1)$$

$$a \mid b \quad \text{und} \quad b \neq 0 \quad \implies \quad |a| \leq |b|. \quad (2)$$

Beweis. (1) Die Voraussetzungen liefern ganze Zahlen q, u mit $b = aq$ und $c = au$; der Ausdruck $kb + hc$ berechnet sich somit zu $aqk + auh = a(qk + uh)$. (2) Wegen $a \mid b$ und $b \neq 0$ gibt es ein q mit $|q| \geq 1$, sodass $b = a \cdot q$ ist. Also gilt $|b| = |aq| = |a| \cdot |q| \geq |a|$. \square

Auf Basis der Teilbarkeitsrelation kann man nun die *Teilermenge* T_a einer ganzen Zahl a , das ist die Menge aller positiven Teiler von a , als $T_a := \{q > 0 : q \mid a\}$ definieren.

2.2 Division mit Resten

Ist eine natürliche Zahlen m nicht durch die natürliche Zahl n teilbar, so lässt die Division m/n einen Rest. Auch diese Beobachtung lässt sich auf ganze Zahlen verallgemeinern.

Satz 2 (Divisionssatz). *Für jede ganze Zahl a und jede ganze Zahl $b > 0$ gibt es eindeutig bestimmte ganze Zahlen, den Quotienten q und den Rest r , sodass $a = qb + r$ und $0 \leq r < b$.*

Beweis. Es sei $R = \{a - qb : q \in \mathbb{Z}\}$ und r das (existierende) kleinste nichtnegative Element dieser Menge. Es ist $r < b$ zu zeigen; zwecks eines Widerspruchs nehmen wir $r \geq b$ an, womit $a - qb \geq b$ und sogleich $a - (q+1)b \geq 0$ folgt. Also war r nicht das kleinste nichtnegative Element aus R . Insgesamt erfüllen q und r also die gestellten Bedingungen.

Nun die Eindeutigkeit: Es existiere \tilde{q} und \tilde{r} mit denselben Eigenschaften wie q und r , also $qb + r = \tilde{q}b + \tilde{r}$, mithin $(q - \tilde{q})b = \tilde{r} - r$ und folglich $b \mid \tilde{r} - r$. Mit (2) gilt $|b| \leq |\tilde{r} - r|$ oder $|\tilde{r} - r| = 0$; mit $|\tilde{r} - r| < b$ wegen $0 \leq r < b$, $0 \leq \tilde{r} < b$ ist $r = \tilde{r}$, mithin $q = \tilde{q}$. \square

Für den Quotienten q der Division a/b notieren wir $\lfloor a/b \rfloor := q$; den Rest r schreiben wir als $a \bmod b := r$. Da für $r = 0$ die Gleichung $a = qb + r$ zu $a = qb$ wird, gilt $b \mid a$ genau dann, wenn $a \bmod b = 0$ ist.

2.3 Der größte gemeinsame Teiler und Teilerfremdheit

Für alle $a \neq 0$ ist die Menge T_a nach (2) durch $|a|$ beschränkt und $1 \in T_a$. Deswegen enthält $T_a \cap T_b$ außer für $a = b = 0$ ein größtes Element, was folgende Definition rechtfertigt.

Definition 3. Der *größte gemeinsame Teiler* zweier ganzer Zahlen a und b ist das größte Element von $T_a \cap T_b$. Formal gilt also $\text{ggT}(a, b) := \max(T_a \cap T_b)$; zudem sei $\text{ggT}(0, 0) := 0$.

In Abb. 3 ist der größte gemeinsame Teiler an zwei Beispielen dargestellt; in (b) tritt der wichtige Sonderfall $\text{ggT}(a, b) = 1$ auf, für den wir sogar eine eigene Notation einführen.

Definition 4. Zwei ganze Zahlen a und b sind genau dann *teilerfremd*, wenn $T_a \cap T_b = \{1\}$, das heißt wenn $\text{ggT}(a, b) = 1$. Wenn a und b teilerfremd sind, notieren wir $a \perp b$.

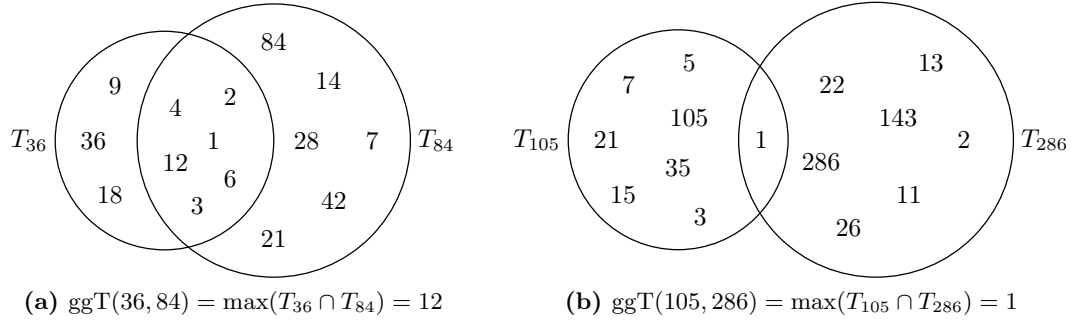


Abb. 3: Venndiagramme zur Veranschaulichung des größten gemeinsamen Teilers

Auf unserer Suche nach einem effizienten Berechnungsverfahren des ggT in Abschnitt 4.1 werden wir einige grundlegende Eigenschaften verwenden, die nun bewiesen werden sollen.

Satz 3 (Eigenschaften des größten gemeinsamen Teilers). *Der größte gemeinsame Teiler genügt für alle ganzen Zahlen a , b und k den Eigenschaften*

$$\text{ggT}(a, ka) = |a|, \quad (\text{Vielfachenregel}) \quad (3)$$

$$\text{ggT}(a, b + ka) = \text{ggT}(a, b) \text{ und} \quad (\text{Reduktionsregel}) \quad (4)$$

$$\text{ggT}(a, b) = \text{ggT}(b, a). \quad (\text{Kommutativgesetz}) \quad (5)$$

Beweis. Der Fall $a = b = 0$ ist trivial, ansonsten gelten folgende Überlegungen. (3) Weil $c \mid a$ auch $c \mid ka$ impliziert, ist $T_a \subseteq T_{ka}$ und deshalb $T_a \cap T_{ka} = T_a$. Mit $\max(T_a) = |a|$ folgt die Behauptung. (4) Mit (1) folgt aus $c \mid a$ und $c \mid b$ auch $c \mid b + ka$ und damit $T_a \cap T_b \subseteq T_a \cap T_{b+ka}$. Außerdem kann man aus $c \mid a$ und $c \mid b + ka$ folgern, dass $a = nc$ und $b + ka = mc$, was insgesamt $b = c(m - kn)$ und somit $c \mid b$ impliziert. Es ist also auch $T_a \cap T_b \supseteq T_a \cap T_{b+ka}$, mithin $T_a \cap T_b = T_a \cap T_{b+ka}$ und $\max(T_a \cap T_b) = \max(T_a \cap T_{b+ka})$. (5) Wegen $T_a \cap T_b = T_b \cap T_a$ ist auch $\max(T_a \cap T_b) = \max(T_b \cap T_a)$. \square

Eine fundamentale Eigenschaft des größten gemeinsamen Teilers der Zahlen a und b ist, dass er sich als Linearkombination von a und b mit ganzzahligen Koeffizienten darstellen lässt. In Abschnitt 4.1 lernen wir ein Verfahren kennen, die Koeffizienten zu berechnen.

Lemma 1 (Lemma von Bézout). *Für alle ganzen Zahlen a und b existieren ganze Zahlen m und n , sodass $\text{ggT}(a, b) = am + bn$.*

Beweis. Die Zahlen a und b seien o. B. d. A.³ positiv. Wir betrachten das kleinste positive Element $d = ax + by$ von $\{am + bn : m, n \in \mathbb{Z}\}$. Nach Satz 2 gibt es ganze Zahlen q und r , sodass $a = qd + r$ mit $0 \leq r < d$, womit $r = a - qd = a - q(ax + by) = a(1 - qx) + b(-qy)$. Es muss also $r = 0$ sein, da ansonsten d nicht minimal wäre, womit $a = qd$, mithin $d \mid a$. Analog erhält man $d \mid b$; d ist also ein gemeinsamer Teiler von a und b . Ist $c \in T_a \cap T_b$, so gilt nach (1) auch $c \mid ax + by = d$, mit (2) ist $c \leq d$, es folgt $\text{ggT}(a, b) = ax + by$. \square

³„ohne Beschränkung der Allgemeinheit“, man meint damit, dass die Allgemeingültigkeit der Aussage trotz der zusätzliche Einschränkung gewahrt bleibt. In diesem Falle ist $a = 0$ oder $b = 0$ trivial und falls z. B. $a < 0$, so zeigt man $\text{ggT}(-a, b) = (-a)m + bn$, womit $\text{ggT}(a, b) = a(-m) + bn$ gilt.

Für teilerfremde ganze Zahlen a und b gilt sogar eine Verschärfung dieses Hilfssatzes.

Lemma 2. *Es seien a und b ganze Zahlen. Sodann gibt es ganze Zahlen m und n mit $am + bn = 1$ genau dann, wenn $a \perp b$.*

Beweis. Die Hinrichtung folgt aus Lemma 1; für die Rückrichtung gelte $am + bn = 1$; mit $d \mid a$ und $d \mid b$ folgt nach (1), dass $d \mid am + bn = 1$, also $d = \pm 1$, womit $a \perp b$. \square

Damit können wir eine wichtige Eigenschaft der Teilerfremdheit beweisen.

Satz 4 (Teilerfremdheit von Produkten). *Für alle ganzen Zahlen a , b und c ist $a \perp b$ und $a \perp c$ äquivalent mit $a \perp bc$.*

Beweis. Nach Voraussetzung und Lemma 2 ist $am + bn = 1$ und $ax + cy = 1$, womit $1 = (am + bn)(ax + cy) = a(amx + mcy + bnx) + bc(ny)$ und damit nach Lemma 2 auch $a \perp bc$. Ist dagegen $am + bcn = 1$, dann folgt $am + b(cn) = 1$ und $am + c(bn) = 1$. \square

2.4 Primzahlen und Primfaktorzerlegung

Von ganz besonderer Bedeutung für die RSA-Verschlüsselung sind, wie wir in der Einleitung bereits festgestellt haben, *Primzahlen*, die folgendermaßen definiert werden.

Definition 5. Eine ganze Zahl $a > 1$, die nur die *trivialen Teiler* ± 1 und $\pm a$ besitzt, für die also $|T_a| = 2$ gilt, nennt man *Primzahl*; ansonsten heißt sie *zusammengesetzt*.

Die Besonderheit von Primzahlen ist, dass jede ganze Zahl $a > 0$ eindeutig in ein Produkt von Primzahlen *faktorisiert* werden kann; sie sind die „Atome“ der Zahlentheorie, was sich in folgendem Hilfssatz ausdrückt.

Lemma 3 (Lemma von Euklid). *Ist p eine Primzahl und sind $a > 0$ sowie $b > 0$ ganze Zahlen mit $p \mid ab$, dann gilt entweder $p \mid a$ oder $p \mid b$.*

Beweis. Aus $p \nmid a$ und $p \nmid b$ folgt nach Definition 5 auch $p \perp a$ und $p \perp b$; mit Satz 4 gilt $p \perp ab$, was im Widerspruch zu $p \mid ab$ steht; deshalb gilt entweder $p \mid a$ oder $p \mid b$. \square

Dieser Hilfssatz ermöglicht es, die Eindeutigkeit der Primfaktorzerlegung zu zeigen.

Satz 5 (Fundamentalsatz der Arithmetik). *Für jede ganze Zahl $a > 0$ existiert eine eindeutige Faktorisierung in Primzahlen, die so genannte kanonische Primfaktorzerlegung*

$$a = p_1^{e_1} \cdot p_2^{e_2} \cdot \dots \cdot p_n^{e_n} = \prod_{k=1}^n p_k^{e_k}, \text{ mit Primzahlen } p_1 < p_2 < \dots < p_n. \quad (6)$$

Beweis. Zuerst beweisen wir mit vollständiger Induktion über a , dass für jede Zahl a eine Primfaktorzerlegung existiert. Für $a = 1$ ist diese Zerlegung offensichtlich⁴. Als Induktionsannahme seien alle k mit $1 \leq k < a$ in Primfaktoren zerlegbar. Für den Induktionsschritt gibt es zwei Fälle: Ist die Zahl a eine Primzahl, so ist der Satz offensichtlich

⁴man kann darüber streiten, ob das „leere Produkt“ der Eins mit $e_1 = 0$ als „Primfaktorzerlegung“ bezeichnet werden darf, da die Eins keine Primzahl ist.

erfüllt; andernfalls gibt es zwei ganze Zahlen $1 \leq m < a$ und $1 \leq n < a$ mit $a = m \cdot n$. Da nach der Induktionsannahme m und n eine Zerlegung besitzen, besitzt auch a eine solche.

Nun ist die Eindeutigkeit der Zerlegung zu zeigen, was erneut mit vollständiger Induktion über a geschehen soll. Der Fall $a = 1$ ist trivial. Ansonsten besitze $a > 1$ die beiden Darstellungen

$$p_1^{e_1} \cdot \dots \cdot p_n^{e_n} = a = q_1^{f_1} \cdot \dots \cdot q_m^{f_m}.$$

Mit $p_1 \mid a$ gilt auch $p_1 \mid q_1^{f_1} \cdot \dots \cdot q_m^{f_m}$, womit es nach Lemma 3 ein k gibt, sodass $p_1 \mid q_k$, also $p_1 = q_k$ ist. Laut Induktionsvoraussetzung stimmen die beiden Darstellungen

$$p_1^{e_1-1} \cdot p_2^{e_2} \cdot \dots \cdot p_n^{e_n} = a/p_1 = q_1^{f_1} \cdot \dots \cdot q_k^{f_k-1} \cdot \dots \cdot q_m^{f_m}$$

bis auf die Reihenfolge überein, da $a/p_1 < a$, womit auch die beiden Darstellungen von a übereinstimmen. \square

3 Kongruenzen und Restklassen

3.1 Definition und Eigenschaften von Kongruenzen und Restklassen

Aufbauend auf dem Divisionssatz aus Abschnitt 2.2 definiert man die *Restklasse* $R_r^m := \{qm + r : q \in \mathbb{Z}\}$, das sind alle Zahlen, die bei der Division durch m den Rest r lassen. In Abb. 4 ist diese Definition an einem Beispiel veranschaulicht. Jede ganze Zahl ist aufgrund der Eindeutigkeit von r auch eindeutig das Element einer bestimmten Restklasse. Sind die Zahlen a und b in der gleichen Restklasse R_r^m , so sagt man, sie sind *kongruent modulo m* . Weil $a - qm = r = b - q'm$ äquivalent zu $a - b = m(q - q')$ ist, gelte folgende Definition.

Definition 6. Zwei ganze Zahlen a und b nennt man *zueinander kongruent modulo m* , wenn die Zahl $m > 0$ ein Teiler der Differenz $a - b$ ist. Man nennt m den *Modul* und notiert $a \equiv b \pmod{m}$. Ist a nicht kongruent b modulo m , so schreibt man $a \not\equiv b \pmod{m}$.

$$a \equiv b \pmod{m} \quad :\Longleftrightarrow \quad m \mid (a - b) \quad \Longleftrightarrow \quad a \bmod m = b \bmod m. \quad (7)$$

Kongruenzen kann man wie Gleichungen behandeln, weil sie, wie wir nun beweisen werden, die Eigenschaften von *Äquivalenzrelationen* (*Reflexivität*, *Symmetrie* und *Transitivität*) besitzen.

Zahl	-4	-3	-2	-1	0	1	2	3	4	5	6	7	8	9	10	11
Rest	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
	$q = -1$				$q = 0$				$q = 1$				$q = 2$			

Abb. 4: Ein Ausschnitt aus der Restklasse $R_1^4 = \{\dots, -3, 1, 5, 9, \dots\}$

Satz 6 (Die Kongruenzbeziehung als Äquivalenzrelation). *Für beliebige ganze Zahlen a, b, c und beliebige Module m gelten die Eigenschaften*

$$a \equiv a \pmod{m} \quad (\text{Reflexivität}) \quad (8)$$

$$a \equiv b \implies b \equiv a \pmod{m} \quad (\text{Symmetrie}) \quad (9)$$

$$a \equiv b \text{ und } b \equiv c \implies a \equiv c \pmod{m} \quad (\text{Transitivität}) \quad (10)$$

Beweis. (8) Die Aussage $a \equiv a \pmod{m}$ ist äquivalent zu $m \mid a - a = 0$. (9) Die Äquivalenz $a \equiv b \pmod{m}$ liefert $m \mid a - b$, woraus nach (1) auch $m \mid -1 \cdot (a - b) = b - a$ folgt, was $b \equiv a \pmod{m}$ impliziert. (10) Aus $a \equiv b \pmod{m}$ und $b \equiv c \pmod{m}$ folgt $m \mid a - b$ und $m \mid b - c$. Addition der beiden rechten Seiten nach (1) liefert $m \mid a - c$. \square

3.2 Rechnen mit Kongruenzen

Die Verwandtschaft zwischen Kongruenzen und Gleichungen geht über die Einhaltung der Äquivalenzaxiome sogar noch hinaus. Beide kann man, wie wir nun zeigen werden, durch verschiedene arithmetische Operationen umformen.

Satz 7 (Rechnen mit Kongruenzen). *Für alle ganzen Zahlen a, b, c, d und beliebige Module m gelten folgende Regeln für die Umformung von Kongruenzen:*

$$a \equiv b \quad \text{und} \quad c \equiv d \quad \implies \quad a + c \equiv b + d \pmod{m} \quad (11)$$

$$a \equiv b \quad \text{und} \quad c \equiv d \quad \implies \quad ac \equiv bd \pmod{m} \quad (12)$$

$$ad \equiv bd \quad \text{und} \quad d \perp m \quad \implies \quad a \equiv b \pmod{m} \quad (13)$$

Beweis. Die Äquivalenz $a \equiv b \pmod{m}$ liefert $m \mid a - b$; $c \equiv d \pmod{m}$ liefert $m \mid c - d$. (11) Mit (1) erhält man daraus $m \mid (a - b) + (c - d) = (a + c) - (b + d)$, was gleichbedeutend mit $a + c \equiv b + d \pmod{m}$ ist. (12) Mit (1) erhält man daraus $m \mid ac - bc$ und $m \mid bc - bd$, was durch erneutes Anwenden von (1) zu $m \mid ac - bd$ wird. (13) Aus $ad \equiv bd \pmod{m}$ folgt $m \mid ad - bd = d(a - b)$. Da aber $d \perp m$ gilt, haben d und m keine Primfaktoren gemeinsam, sie sind somit alle in $a - b$ zu finden, womit $m \mid a - b$ und damit $a \equiv b \pmod{m}$. \square

Da also jede Zahl $a \in R_r^m$, ein so genannter *Repräsentant* aus R_r^m , bei der Addition und Multiplikation (und damit auch die Basis bei der Exponentiation) durch eine andere Zahl $b \in R_r^m$ ersetzt werden kann, können während der Berechnung immer alle Werte *a modulo m reduziert*, das heißt durch $a \bmod m$ ersetzt werden, weshalb die Kongruenzrechnung bei der RSA-Verschlüsselung so wichtig ist (siehe Abschnitt 4.2).

3.3 Verändern des Moduls

Zuletzt lernen wir zwei Regeln kennen, die es ermöglichen, Kongruenzen mit verschiedenen Modulen zu kombinieren, bzw. eine Kongruenz aufzuspalten.

Satz 8 (Verändern des Moduls). *Für alle ganzen Zahlen a und b sowie beliebige Module m und n gelten die Regeln*

$$a \equiv b \pmod{m}, a \equiv b \pmod{n} \quad \text{und} \quad m \perp n \quad \implies \quad a \equiv b \pmod{mn}, \quad (14)$$

$$a \equiv b \pmod{m}, a \equiv b \pmod{n} \quad \iff \quad a \equiv b \pmod{mn}. \quad (15)$$

Beweis. (14) Aus $a \equiv b \pmod{m}$ folgt $m \mid a - b$, aus $a \equiv b \pmod{n}$ folgt $n \mid a - b$. Die Primfaktorzerlegung von $a - b$ enthält also die Primfaktoren von m und n . Da $m \perp n$ haben m und n keine gemeinsamen Primfaktoren, $a - b$ enthält also sogar die Primfaktoren von mn , es gilt $mn \mid a - b$ und damit $a \equiv b \pmod{mn}$. (15) Aus $a \equiv b \pmod{mn}$ folgt $mn \mid a - b$, also $a - b = mnq$ und damit auch $m \mid a - b$ und $n \mid a - b$. \square

4 Zahlentheoretische Algorithmen

4.1 Der (erweiterte) Algorithmus von Euklid

Haben wir in Abschnitt 2.3 den größten gemeinsamen Teiler d zweier ganzer Zahlen a und b nur theoretisch betrachtet, so lernen wir jetzt den Algorithmus von Euklid kennen, mit dessen Hilfe man d berechnen kann.

Das Verfahren funktioniert folgendermaßen: Man bestimmt q und r , sodass $a = qb + r$ mit $0 \leq r < b$ nach Satz 2 erfüllt ist. Ist $r = 0$, so ist man wegen (3) schon fertig, denn $\text{ggT}(qb, b) = |b|$. Ansonsten gilt wegen $r = a - qb$ sowie mit (5) und (4), dass

$$\text{ggT}(a, b) = \text{ggT}(b, a) = \text{ggT}(b, a - qb) = \text{ggT}(b, r). \quad (16)$$

War ursprünglich $a > b$, so ist jetzt $a > b > r$. Zur Berechnung des „einfacheren“ $\text{ggT}(b, r)$ verfährt man wie zuvor.

Um den Algorithmus zu verdeutlichen wollen wir $\text{ggT}(2478, 777)$ bestimmen. Mit $a = 2478$ und $b = 777$ liefert sukzessives Berechnen von $a = qb + r$, dass

$$\begin{aligned} 2478 &= 3 \cdot 777 + 147, & \text{mit } a_1 = 777 \text{ und } b_1 = 147 \text{ erhält man} \\ 777 &= 5 \cdot 147 + 42, & \text{mit } a_2 = 147 \text{ und } b_2 = 42 \text{ erhält man} \\ 147 &= 3 \cdot 42 + 21, & \text{wegen } a_3 = 2 \cdot b_3 \text{ ist } d = 21. \end{aligned} \quad (17)$$

Es zeigt sich aber, dass man häufig gar nicht d sucht, sondern die Koeffizienten m und n aus Lemma 1, sodass $d = am + bn$. Wie kann man das erreichen? In (17) steht die Lösung schon fast da, löst man nämlich jede der Gleichungen nach dem Rest r auf, so erhält man durch fortgesetztes Einsetzen

$$\begin{aligned} a &= 3b + 147 & \text{liefert} & & 147 &= a - 3b \\ b &= 5 \cdot 147 + 42 & \text{liefert} & & 42 &= b - 5 \cdot (a - 3b) = -5a + 16b \\ 147 &= 3 \cdot 42 + 21 & \text{liefert} & & 21 &= (a - 3b) - 3 \cdot (-5a + 16b) \\ & & & & &= 16a - 51b. \end{aligned} \quad (18)$$

Auf dieser Beobachtung basiert ein Verfahren, das man den „erweiterten“ Algorithmus von Euklid nennt. Es berechnet in jedem Schritt k außer a_k und b_k immer vier Variablen u_k , v_k , m_k und n_k , sodass

$$a_k = u_k a + v_k b \quad \text{und} \quad b_k = m_k a + n_k b. \quad (19)$$

Bevor wir daran gehen, das Verfahren in seiner vollen Allgemeinheit zu beschreiben, soll am Beispiel $a = 2478$ und $b = 777$ gezeigt werden, wie die Koeffizienten berechnet werden (siehe Tab. 1). Die Spalten a , b , q und r sollten jetzt keine Probleme mehr machen; die Werte von u , v , m und n kann man einfach aus (18) ablesen und mit (19) sogar die allgemeine Formel daraus ableiten, wegen $a_{k+1} = b_k$ und $b_{k+1} = a_k - q_k b_k$ gilt $u_{k+1} = m_k$ und $v_{k+1} = n_k$ sowie $m_{k+1} = u_k - q_k m_k$ und $n_{k+1} = v_k - q_k n_k$.

Der gesamte Algorithmus ist im Flussdiagramm in Abb. 5 zusammengefasst und mit Invarianten versehen, die wir im nun folgenden Korrektheitsbeweis verwenden werden. Die Variablen a_k und b_k sind dort mit x und y bezeichnet.

k	u_k	v_k	m_k	n_k	a_k	b_k	q_k	r_k
0	1	0	0	1	2478	777	3	147
1	0	1	1	-3	777	147	5	42
2	1	-3	-5	16	147	42	3	21
3	-5	16	16	-51	42	21	2	0

Tab. 1: Der erweiterte Algorithmus von Euklid am Beispiel $a = 2478$ und $b = 777$

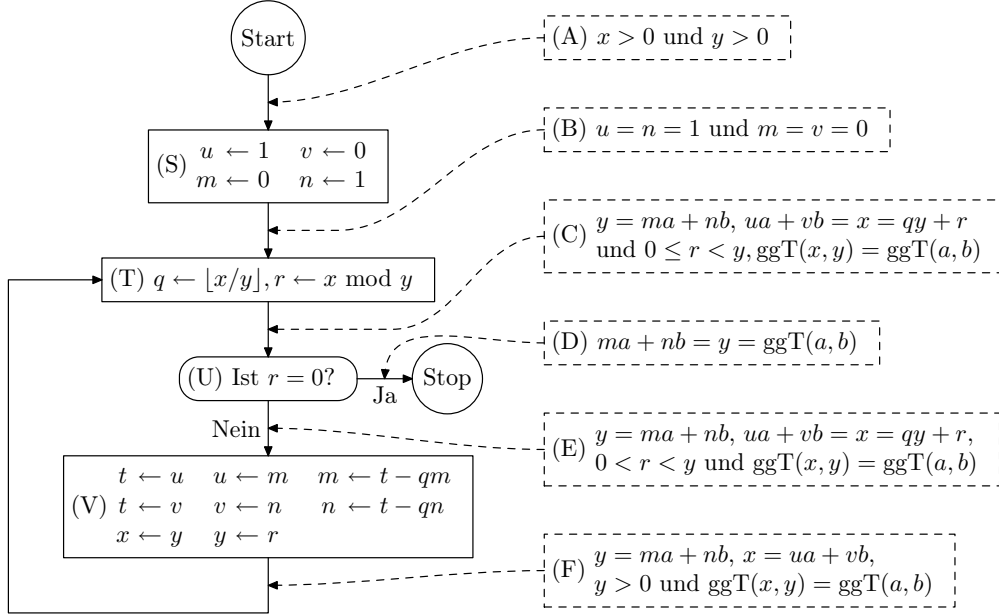


Abb. 5: Flussdiagramm und Invarianten des erweiterten Euklidischen Algorithmus zur Berechnung von $\text{ggT}(x, y)$ mit $x > 0$ und $y > 0$

Satz 9. Der erweiterte euklidische Algorithmus berechnet, zwei ganze Zahlen $x = a > 0$ und $y = b > 0$ gegeben, das Tripel (d, m, n) mit $d = \text{ggT}(a, b) = am + bn$.

Beweis. Wir beweisen den Satz mit vollständiger Induktion über der Anzahl der abgearbeiteten Schritte, indem wir zeigen, dass in jedem Schritt die Gültigkeit aller Vorbedingungen in Abb. 5 auch die Gültigkeit aller Nachbedingungen impliziert.

Bedingung (A) ist als Induktionsanfang nach Voraussetzung erfüllt. Nach Schritt (S) gilt trivialerweise (B). Durch Einsetzen lässt sich $(B) \Rightarrow (F)$ zeigen, womit es genügt, zu zeigen, dass bei gültigem (F) nach Schritt (T) auch (C) gilt, was offensichtlich aus Satz 2 folgt. Gleichung (3) liefert $\text{ggT}(a, b) = \text{ggT}(x, y) = \text{ggT}(qy, y) = |y|$, womit nach Schritt (U) unter Voraussetzung (C) mit $r = 0$ die Nachbedingung (D) folgt. Ist $r \neq 0$, so gilt trivialerweise (E). Nun kommt der einzig nicht-triviale Teil des Beweises, die Korrektheit von (F) nach Schritt (V) mit der Vorbedingung (E). Aus (E) erhält man

$$y = ma + nb \quad (\text{I}), \quad x = ua + vb \quad (\text{II}) \quad \text{und} \quad x = qy + r \quad (\text{III}).$$

Löst man Gleichung (III) nach r auf, so erhält man mit (II) und (I) den Zusammenhang

$$\begin{aligned} r &= x - qy \stackrel{\text{(II)}}{=} ua + vb - qy \stackrel{\text{(I)}}{=} ua + vb - q(ma + nb) \\ &= (u - qm) \cdot a + (v - qn) \cdot b. \end{aligned}$$

Nach den Substitutionen $m \leftarrow u - qm$, $n \leftarrow v - qn$ und $y \leftarrow r$ aus Schritt (V) gilt also $y = ma + nb$. Aus $y = ma + nb$ in (E) erhält man nach $u \leftarrow m$, $v \leftarrow n$ und $x \leftarrow y$ auch $x = ua + vb$. Die Nachbedingung $y > 0$ ist wegen $y \leftarrow r$ und $r > 0$ klar, $\text{ggT}(a, b) = \text{ggT}(x, y) = \text{ggT}(y, x - qy) = \text{ggT}(y, r)$ folgt mit (4).

Da vor (V) immer $0 < r < y < x$ ist und in (V) $x \leftarrow y$, $y \leftarrow r$ gesetzt wird, berechnet der Algorithmus in endlich vielen Schritten das richtige Ergebnis. \square

4.2 Die modulare Exponentiation

Ein weiteres Problem, das bei der RSA-Verschlüsselung auftritt, besteht darin, $a^b \bmod m$ zu bestimmen. Auf den ersten Blick scheint das trivial zu sein: Warum nicht einfach a^b berechnen, und dann modulo m reduzieren? Später werden wir aber mit a , b der Größenordnung $\gg 10^3$ arbeiten, damit ist a^b von der Größenordnung $\gg 10^{3000}$. Bei „nur“ ungefähr 10^{78} Atomen im Universum ist dieser naive Ansatz kaum praktikabel.

Es gibt stattdessen einen cleveren Algorithmus, für sehr große Zahlen a und b in wenigen Millisekunden $a^b \bmod m$ zu berechnen, die *modulare Exponentiation*. Die Kernidee des Verfahrens ist schnell geschildert. Es sei $(b_n b_{n-1} \dots b_0)_2$ die Binärdarstellung von b . Sodann gilt mit den Potenzgesetzen, dass

$$a^b = \prod_{k=0}^n (a^{2^k})^{b_k}, \quad \text{wegen } b = \sum_{k=0}^n b_k \cdot 2^k.$$

Da die Koeffizienten b_k entweder 0 oder 1 sind, genügt es also, diejenigen Basen a^{2^k} miteinander zu multiplizieren, für die $b_k = 1$. Aufgrund der Identität

$$a^{2^{k+1}} = (a^{2^k})^2 \tag{20}$$

kann man die Basen durch fortgesetzte Quadrierung modulo m sehr schnell berechnen. Das Verfahren soll anhand des Beispiels $a = 8$, $b = 91$ und $m = 11$ näher erläutert werden. Mit $b = (1011011)_2$ und $8^2 = 64 \equiv 9 \pmod{11}$, $9^2 = 81 \equiv 4 \pmod{11}$, $4^2 = 16 \equiv 5 \pmod{11}$, $5^2 = 25 \equiv 3 \pmod{11}$, $3^2 = 9$ sowie $9^2 = 81 \equiv 4 \pmod{11}$ gilt

k	6	5	4	3	2	1	0	
b_k	1	0	1	1	0	1	1	
a^{2^k}	4	9	3	5	4	9	8	$\pmod{11}$.

Wegen (12) lautet das Ergebnis $a^b \bmod m = 4 \cdot 3 \cdot 5 \cdot 9 \cdot 8 = 540 \cdot 8 \equiv 1 \cdot 8 \pmod{11}$.

In Abb. 6 ist das Flussdiagramm des Verfahrens in seiner allgemeinen Form abgebildet. Wir werden nun anhand der eingezeichneten Invarianten einen Korrektheitsbeweis der modularen Exponentiation führen.

Satz 10. Die modulare Exponentiation berechnet für alle ganze Zahlen a und b , wobei $b \geq 0$ und nicht $a = b = 0$, sowie jeden Modul m eine Zahl d , sodass $d = a^b \bmod m$ gilt.

Beweis. Der Induktionsanfang (A) gilt nach Voraussetzung. Die Korrektheit der Nachbedingung (B) nach (S) ist offensichtlich. Wir betrachten Schritt (T). Weil $k = n$ mit $k = -1$ im Widerspruch steht, genügt es, die Korrektheit von Nachbedingung (C) aus der Korrektheit von (G) zu folgern, wegen $d \equiv a^{(b_n b_{n-1} \dots b_{k+1})_2} \pmod{m}$ nach (G) folgt mit $k = -1$ auch $d \equiv a^b \pmod{m}$. Die Korrektheit von (D) ist offensichtlich. Mit⁵

$$(a^{(b_n b_{n-1} \dots b_{k+1})_2})^2 = a^{2 \cdot (b_n b_{n-1} \dots b_{k+1})_2} \stackrel{(21)}{=} a^{(b_n b_{n-1} \dots b_{k+1} 0)_2}$$

folgt für (U) dann (E) aus (D). (F) kann man mit einer Fallunterscheidung beweisen. Für $b_k = 0$ ist (F) ja schon aus (E) erfüllt, für $b_k = 1$ gilt⁶

$$a^{(b_n b_{n-1} \dots b_{k+1} 0)_2} \cdot a = a^{(b_n b_{n-1} \dots b_{k+1} 0)_2 + 1} \stackrel{(22)}{=} a^{(b_n b_{n-1} \dots b_{k+1} b_k)_2}.$$

Zuletzt folgt (G) aus (F) durch simple Umbenennung der Variable k . Der Algorithmus bricht irgendwann ab, weil k nur die Werte aus $\{n, n-1, \dots, -1\}$ annimmt. \square

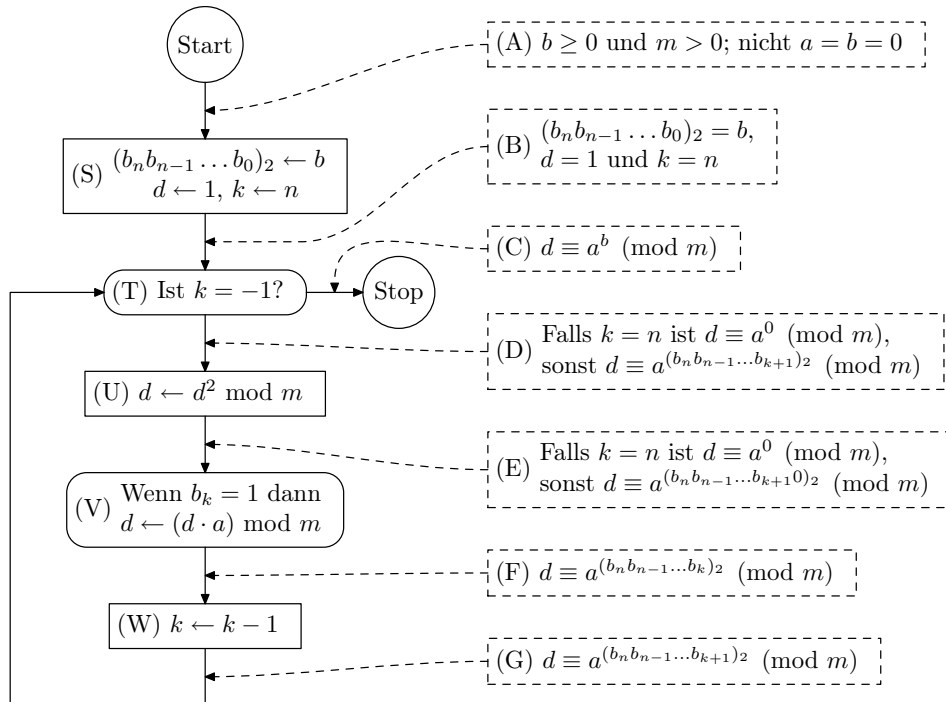


Abb. 6: Flussdiagramm und Invarianten der modularen Exponentiation, die für $b \geq 0$ und $m > 0$ den Ausdruck $a^b \bmod m$ berechnet, wenn nicht zugleich $a = 0$ und $b = 0$

⁵ $2 \cdot (b_n \dots b_1 b_0)_2 = b_n \cdot 2^{n+1} + \dots + b_0 \cdot 2^1 + 0 \cdot 2^0 = (b_n \dots b_0 0)_2$ (21)

⁶ $(b_n \dots b_1 0)_2 + 1 = b_n \cdot 2^n + \dots + b_1 \cdot 2^1 + 1 \cdot 2^0 = (b_n \dots b_0 1)_2$ (22)

4.3 Das Lösen modularer Gleichungen

4.3.1 Das Finden von multiplikativen Inversen

Ein Problem bei der Suche nach einem RSA-Schlüsselpaar besteht darin, eine ganzzahlige Lösung x zu finden, sodass

$$ax \equiv 1 \pmod{m}. \quad (23)$$

Satz 11. Für jede ganze Zahl a und jeden Modul m mit $a \perp m$ gibt es eine eindeutig bestimmte ganze Zahl x modulo m sodass $ax \equiv 1 \pmod{m}$.

Beweis. Wegen $a \perp m$ gibt es nach Lemma 2 zwei ganze Zahlen x und y , sodass $ax + my = 1$, damit ist $ax - 1 = -my$ und $m \mid ax - 1$ ist erfüllt, womit $ax \equiv 1 \pmod{m}$. Eindeutigkeit: Sei $x \not\equiv \tilde{x} \pmod{m}$ mit $ax \equiv 1 \equiv a\tilde{x} \pmod{m}$, dann wäre $x \equiv x a \tilde{x} \equiv \tilde{x} \pmod{m}$. \square

In der Praxis zieht man zur Berechnung von x den erweiterten Algorithmus von Euklid (siehe Abschnitt 4.1) heran. Man nennt x das *multiplikative Inverse* („Kehrwert“) zu a .

4.3.2 Der chinesische Restsatz

Da es für prime Module p zu jedem $a \not\equiv 0 \pmod{p}$ ein multiplikatives Inverses gibt, sind Kongruenzen modulo einer Primzahl für manche Beweise besonders hilfreich; wir betrachten einen Satz, mit dem man viele Kongruenzen auf diesen Fall zurückführen kann.

Definition 7. Sei der Modul m das Produkt paarweise teilerfremder Modulen m_1, m_2, \dots, m_n . Ein Tupel (a_1, a_2, \dots, a_n) heißt *chinesischer Rest* einer ganzen Zahl a modulo m , wenn $a \equiv a_1 \pmod{m_1}$, $a \equiv a_2 \pmod{m_2}$, \dots , $a \equiv a_n \pmod{m_n}$.

Satz 12 (Der chinesische Restsatz). Jeder Zahl a modulo $m = m_1 m_2 \dots m_n$ kann eindeutig ihr chinesischer Rest (a_1, a_2, \dots, a_n) zugeordnet werden und umgekehrt.

Beweis. Zuerst geben wir die Zuordnung $(a_1, a_2, \dots, a_n) \mapsto a$ an. Für $1 \leq k \leq n$ sei $\mu_k = m/m_k$, womit wegen m_k paarweise teilerfremd auch $\mu_k \perp m_k$ gilt. Nach Satz 11 existieren also eindeutige Zahlen x_k mit $\mu_k x_k \equiv 1 \pmod{m_k}$. Nun sei $a = \sum_{k=1}^n a_k \mu_k x_k$ (per Konstruktion eindeutig). Offensichtlich ist $a_k \mu_k x_k \equiv a_k \cdot 1 \equiv a_k \pmod{m_k}$. Für $k \neq j$ gilt $m_j \mid \mu_k$, womit $a_k \mu_k x_k \equiv a_k \cdot 0 \cdot x_k \equiv 0 \pmod{m_j}$ ist, man erhält insgesamt

$$\begin{aligned} a &= a_1 \mu_1 x_1 + \dots + a_k \mu_k x_k + \dots + a_n \mu_n x_n \\ &\equiv 0 + \dots + a_k + \dots + 0 = a_k \pmod{m_k}. \end{aligned}$$

Die Umkehrabbildung $a \mapsto (a_1, a_2, \dots, a_n)$ ist mit $a_k = a \pmod{m_k}$ für $1 \leq k \leq n$ gegeben und aufgrund der Eindeutigkeit von Resten nach Satz 2 eindeutig. \square

Die Darstellung als chinesische Reste werden komponentenweise addiert und multipliziert.

Satz 13 (Rechnen mit chinesischen Resten). Sei $m = m_1 \dots m_n$ mit m_k paarweise teilerfremd und sei $0 \leq a_k < m_k$ sowie $0 \leq b_k < m_k$ für alle $1 \leq k \leq n$; dann gelten für die chinesischen Reste von $a + b$ und $a \cdot b$ die Beziehungen

$$\begin{aligned} (a_1, a_2, \dots, a_n) + (b_1, b_2, \dots, b_n) &= (a_1 + b_1, a_2 + b_2, \dots, a_n + b_n) \text{ und} \\ (a_1, a_2, \dots, a_n) \cdot (b_1, b_2, \dots, b_n) &= (a_1 \cdot b_1, a_2 \cdot b_2, \dots, a_n \cdot b_n). \end{aligned}$$

Beweis. Es sei $a = (a_1, a_2, \dots, a_n)$ und $b = (b_1, b_2, \dots, b_n)$, sodass $a \equiv a_k \pmod{m_k}$ und $b \equiv b_k \pmod{m_k}$ für $1 \leq k \leq n$. Für $c = a + b$ gibt es nach Satz 12 Zahlen c_k mit $c \equiv c_k \pmod{m_k}$. Mit (11) ist $c = a + b \equiv a_k + b_k = c_k \pmod{m_k}$. Analog $a \cdot b$ mit (12). \square

5 Verschlüsselung mit dem RSA-Algorithmus

5.1 Der kleine Satz von Fermat

Eine zentrale Rolle, sowohl für das RSA-Verfahren als auch für die Primzahlerzeugung, besitzt folgender Satz.

Satz 14 (Der kleine Satz von FERMAT). *Für alle ganzen Zahlen a und alle Primzahlen p mit $a \perp p$ gilt der Zusammenhang*

$$a^{p-1} \equiv 1 \pmod{p}. \quad (24)$$

Beweis. Wir betrachten die Zahlen $0, a, 2a, 3a, \dots, (p-1)a$, die paarweise inkongruent modulo p sind, da aus $a \perp p$ und $ma \equiv na \pmod{p}$ mit (13) auch $m \equiv n \pmod{p}$ folgen würde. Insgesamt sind diese Zahlen modulo p betrachtet also eine Permutation der Zahlen $0, 1, 2, \dots, p-1$. Mit (12) gilt also

$$1 \cdot 2 \cdot \dots \cdot (p-1) \equiv a \cdot 2a \cdot \dots \cdot (p-1)a \pmod{p}.$$

Da die Zahlen $1, 2, \dots, p-1$ teilerfremd zu p sind, darf man nach (13) durch sie dividieren und erhält $1 \equiv a^{p-1} \pmod{p}$. \square

5.2 Berechnung der Schlüssel und Verschlüsselung

Wir haben nun alle Hilfsmittel zusammen, um die Frage nach der Konstruktion der Einwegfunktion aus der Einleitung zu beantworten. RON RIVEST, ADI SHAMIR und LEN ADLEMAN setzen folgende Methode zur Berechnung der RSA-Schlüssel an.

1. Wähle zufällig zwei verschiedene Primzahlen p und q .
2. Bilde das Produkt $n = pq$ und berechne die Zahl $k = (p-1)(q-1)$.
3. Bestimme zwei Zahlen d und e , sodass $e \perp k$ und $d \cdot e \equiv 1 \pmod{k}$.
4. Das Paar $O = (e, n)$ ist der *öffentliche RSA-Schlüssel*, der freigegeben werden kann; das Paar $P = (d, n)$ ist der *private RSA-Schlüssel*, der geheimgehalten wird.

Die Funktion zum „Verschlüsseln“ der Nachricht m mit dem Schlüssel S ist definiert als

$$f_S(m) = m^a \bmod n, \quad \text{mit } (a, n) = S. \quad (25)$$

Satz 15 (Korrektheit des RSA-Verfahrens). *Für ein Schlüsselpaar, bestehend aus dem privaten Schlüssel P und dem öffentlichen Schlüssel O , das mit obigem Verfahren konstruiert wurde, gilt der Zusammenhang $f_P \circ f_O = \text{id} = f_O \circ f_P$.*

Beweis. Nach Gleichung (25) gilt mit $O = (e, n)$ und $P = (d, n)$ für das Ergebnis der „Verschlüsselung“ $f_P(f_O(m)) \equiv (m^e)^d = m^{ed} = (m^d)^e \equiv f_O(f_P(m)) \pmod{n}$.

Um die Korrektheit des Verfahrens zu zeigen, müssen wir also die Kongruenz $m^{ed} \equiv m \pmod{n}$ beweisen. Mit $d \cdot e \equiv 1 \pmod{k}$ und (15) gilt wegen $k = (p-1)(q-1)$, dass

$$ed \equiv 1 \pmod{p-1} \iff p-1 \mid ed-1 \iff ed-1 = t \cdot (p-1). \quad (26)$$

Die letzte Gleichung von (26) können wir nun wieder in die linke Seite der Kongruenz $m^{ed} \equiv m \pmod{n}$ einsetzen und erhalten

$$m^{ed} = m^{t(p-1)+1} = m \cdot (m^{p-1})^t. \quad (27)$$

Als erstes wollen wir die Kongruenz $m^{ed} \equiv m \pmod{p}$ beweisen. Im Falle $p \mid m$ gilt $m^{ed} \equiv 0 \equiv m \pmod{p}$; ansonsten wird mit (27) und Satz 14 klar, dass $m^{p-1} \equiv 1 \pmod{p}$, also $m \cdot (m^{p-1})^t \equiv m \cdot 1^t \equiv m \pmod{p}$ gilt. Analog dazu kann man auf die Kongruenz $m^{ed} \equiv m \pmod{q}$ folgern. Wegen $p \perp q$ gilt also mit (14), dass $m^{ed} \equiv m \pmod{n}$. \square

6 Primzahltests und Primzahlerzeugung

Wie sich in Abschnitt 5.2 herausgestellt hat sind zur Berechnung von RSA-Schlüsseln sehr große Primzahlen nötig, weshalb wir nach einer Methode suchen müssen, diese zu erzeugen.

6.1 Probedivision und Sieb des Eratosthenes

Wenn man eine natürliche Zahl n „per Hand“ darauf untersuchen will, ob sie zusammengesetzt ist oder nicht, kann man folgenden Satz zur Hilfe nehmen.

Satz 16. *Ist $n > 1$ eine zusammengesetzte natürliche Zahl, dann hat n einen Primteiler p mit $1 < p \leq \sqrt{n}$.*

Beweis. Mit Satz 5 gibt es eine kleinste Primzahl p , sodass $n = pa$. Da n zusammengesetzt ist, hat n noch mindestens einen weiteren Primteiler, womit $p \leq a$ gilt. Wir nehmen $\sqrt{n} < p \leq a$ an, damit ist $n = pa > n$, ein Widerspruch. \square

Um festzustellen, ob n zusammengesetzt oder prim ist, muss man also nur für alle Primzahlen p mit $1 < p \leq \sqrt{n}$ prüfen, ob $p \mid n$. Ist das nicht der Fall, so ist n prim. Die Primzahlen für diese so genannte *Probedivision* kann man mit dem *Sieb des Eratosthenes* bestimmen, das nun beschrieben werden soll.

Man beginnt damit, die Zahlen k mit $2 \leq k \leq n$ niederzuschreiben. Nun wird die kleinste Zahl betrachtet und alle ihre Vielfache ausgestrichen. Man setzt das Verfahren nun immer wieder mit der nächstgrößeren Zahl, die noch nicht weggestrichen ist, fort, bis man alle Zahlen behandelt hat. Übrig bleiben alle Zahlen p , die keinen Teiler in der Menge $\{2, 3, \dots, p-1\}$ haben, also Primzahlen sind (siehe Abb. 7).

Für sehr große Zahlen ist es aber extrem zeitaufwendig, das Sieb des Eratosthenes anzuwenden. Nach unserer Grundannahme aus der Einleitung sind Verfahren, die auch

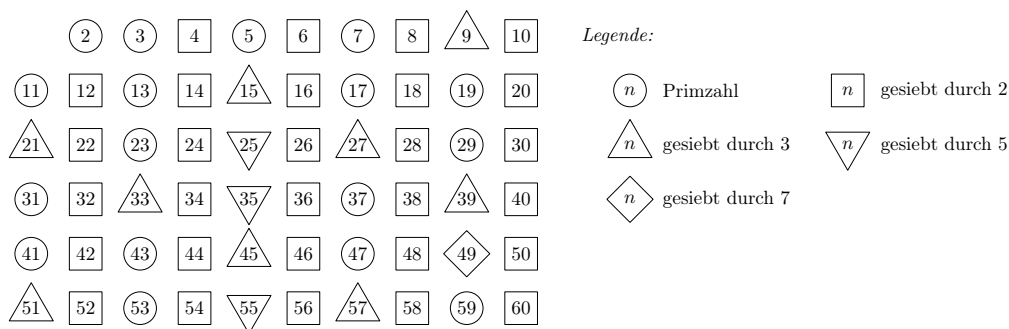


Abb. 7: Beispiel für die Funktionsweise des Siebs von Eratosthenes mit $n = 60$

eine Faktorisierung der gefundenen Zahl „mitliefern“, ungeeignet, große Primzahlen zu suchen, worauf die Sicherheit von RSA ja gerade basiert. Stattdessen erzeugt man mit dem Sieb eine Primzahlliste bis etwa 10^6 , um die Zahlen, die man testet, mit der Probedivision zu untersuchen, bevor man „richtige“ Primzahltests benutzt.

6.2 Der Fermat-Test

Für große Zahlen verwendet man stattdessen Tests, die nachweisen können, dass eine Zahl mit hoher Wahrscheinlichkeit eine Primzahl ist, zum Beispiel den *Fermat-Test*.

Durch den kleinen Satz von FERMAT haben wir die Möglichkeit, zu zeigen, dass eine natürliche Zahl n zusammengesetzt ist. Man berechnet mithilfe der modularen Exponentiation aus Abschnitt 4.2 für ein a mit $n \nmid a$ den Rest $y = a^{n-1} \bmod n$. Ist $y \neq 1$, so ist n nach Satz 14 wegen $a^{n-1} \not\equiv 1 \pmod{n}$ keine Primzahl. Ist aber $y = 1$, so kann man keine definitive Aussage machen, ob n nun Primzahl oder zusammengesetzt ist. Zum Beispiel gilt für $341 = 31 \cdot 11$, dass $2^{340} \equiv 1 \pmod{341}$, wie man mit $340 = (101010100)_2$ leicht nachrechnet.

k	8	7	6	5	4	3	2	1	0
b_k	1	0	1	0	1	0	1	0	0
a^{2^k}	64	256	16	4	64	256	16	4	2.

Wegen $1024 \equiv 1 \pmod{341}$ gilt somit $64 \cdot 16 \cdot 64 \cdot 16 = 1024^2 \equiv 1 \pmod{341}$. Während 341 im Fermat-Test aber wenigstens für $a = 3$ als zusammengesetzt erkannt wird, gibt es Zahlen, deren Zusammengesetztheit für kein a erkannt wird, die CARMICHAEL-Zahlen.

Die Situation ist in Abb. 8 zusammengefasst. Wenn n eine Primzahl ist, dann erkennt der Fermat-Test dies zuverlässig. Ist n aber zusammengesetzt, so kann er sich irren. Wurde der Test mit der Zahl a als Basis durchgeführt und $a^{n-1} \not\equiv 1 \pmod{n}$, so nennt man a einen *Fermat-Zeugen* für die Zusammengesetztheit von n . Ist jedoch, obwohl n zusammengesetzt ist, $a^{n-1} \equiv 1 \pmod{n}$, so nennt man a einen *Fermat-Lügner*.

Definition 8. Eine zusammengesetzte natürliche Zahl n heißt CARMICHAEL-Zahl, wenn für alle Zahlen a mit $a \perp n$ die Beziehung $a^{n-1} \equiv 1 \pmod{n}$ gilt, jedes solche a also Fermat-Lügner für n ist.

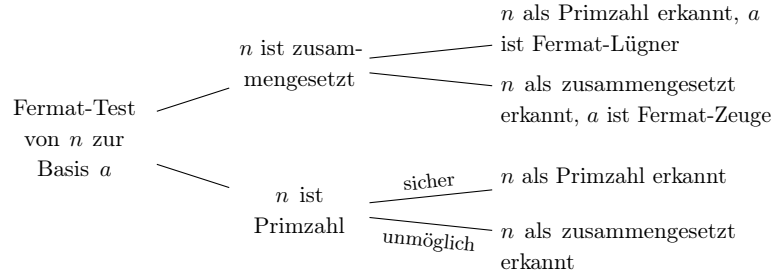


Abb. 8: Überblick über die Ergebnisse eines Fermat-Tests

Die kleinste CARMICHAEL-Zahl ist $561 = 3 \cdot 11 \cdot 17$, da nach dem kleinen Satz von Fermat für jedes $a \perp n$ die Kongruenzen $a^2 \equiv 1 \pmod{3}$, $a^{10} \equiv 1 \pmod{11}$ und $a^{16} \equiv 1 \pmod{17}$ gelten. Daraus erhält man $a^{560} = (a^2)^{280} \equiv 1 \pmod{3}$, $a^{560} = (a^{10})^{56} \equiv 1 \pmod{11}$ und $a^{560} = (a^{16})^{35} \equiv 1 \pmod{17}$, was man nach (14) zu $a^{560} \equiv 1 \pmod{561}$ für alle $a \perp n$ kombinieren kann. Wir führen nun eine notwendige Bedingung für CARMICHAEL-Zahlen an.

Satz 17. *Ist n eine CARMICHAEL-Zahl, so gibt es keine Primzahl p mit $p^2 \mid n$.*

Beweis. Wir nehmen an, es existiere eine Primzahl p mit $p^2 \mid n$. Es sei $n = p^k q$, sodass $k > 1$ und $p \perp q$. Ziel ist es, ein a zu konstruieren, sodass $a \perp n$ und $a^{n-1} \not\equiv 1 \pmod{n}$, womit n keine CARMICHAEL-Zahl wäre. Mit dem chinesischen Restsatz finden wir ein a , sodass $a \equiv p+1 \pmod{p^k}$ und $a \equiv 1 \pmod{q}$. Nach⁷ Lemma 4 ist $a \perp p^k$ und $a \perp q$; mit Satz 4 ist $a \perp n$. Wir wollen nun beweisen, dass $a^{n-1} \not\equiv 1 \pmod{n}$. Klar ist

$$(p+1)^p = 1 + \binom{p}{1}p + \binom{p}{2}p^2 + \cdots + \binom{p}{p}p^p,$$

womit $(p+1)^p \equiv 1 \pmod{p^2}$ gilt. Wegen $a \equiv p+1 \pmod{p^k}$, also nach (15) auch $a \equiv p+1 \pmod{p^2}$, ist also $a^p \equiv 1 \pmod{p^2}$ und daraus $a^n \equiv 1 \pmod{p^2}$. Insbesondere ist damit $a^n \not\equiv a \pmod{p^2}$, was wir nun nutzen können, um $a^{n-1} \equiv 1 \pmod{n}$ zum Widerspruch zu führen. Mit $a^{n-1} \equiv 1 \pmod{n}$ würde nämlich nach (12) und (15) die Kongruenz $a^n \equiv a \pmod{p^2}$ folgen. \square

Nachfolgend soll die Qualität des Tests untersucht werden, wenn n keine CARMICHAEL-Zahl ist. Dazu ist es nötig, einen Hilfssatz einzuführen.

Lemma 5. *Sei $b > 0$. Existiert eine ganze Zahl y mit $y^b \equiv c \pmod{m}$ und $y \perp m$, so haben die Kongruenzen $x^b \equiv c \pmod{m}$ und $x^b \equiv 1 \pmod{m}$ die gleiche Anzahl von Lösungen x .*

⁷**Lemma 4.** *Für alle ganzen Zahlen a, b und alle Module m mit $a \equiv b \pmod{m}$ und $a \perp m$ gilt $b \perp m$.*

Beweis. Es folgt $m \mid a - b$ und damit $a - b = qm$. Wir nehmen an, es existiert ein d mit $dm' = m$ und $db' = b$, womit $a - db' = qdm'$, also $a = d(b' + qm)$; Mit $d \mid m$ und $d \mid a$: Widerspruch zu $a \perp m$. \square

Beweis. Wegen $y \perp m$ gibt es nach Satz 11 ein z mit $yz \equiv 1 \pmod{m}$. Die Funktion $s \mapsto s \cdot z$ ordnet jeder Lösung von $x^b \equiv c \pmod{m}$ eine eindeutige Lösung von $x^b \equiv 1 \pmod{m}$ zu, da $(sz)^b = s^b z^b \equiv cz^b \equiv y^b z^b \equiv (yz)^b \equiv 1 \pmod{m}$; die Umkehrabbildung $w \mapsto w \cdot y$ ordnet außerdem jeder Lösung von $x^b \equiv 1 \pmod{m}$ eine eindeutige Lösung von $x^b \equiv c \pmod{m}$ zu, weil $(wy)^b = w^b y^b \equiv y^b \equiv c \pmod{m}$. \square

Mit wenig Aufwand kann man nun eine untere Schranke $1/2$ für die Wahrscheinlichkeit, dass ein zufällig gewähltes a als Zeuge für die Zusammengesetztheit einer Zahl n (die keine CARMICHAEL-Zahl ist) dient, nachweisen.

Satz 18. *Hat eine zusammengesetzte Zahl $n > 2$ mindestens einen Fermat-Zeugen a mit $a \perp n$, so sind mindestens die Hälfte der Zahlen $1, 2, \dots, n-1$ auch Fermat-Zeugen für n .*

Beweis. Nach Voraussetzung ist $a \perp n$ und es gibt ein a mit $a^{n-1} \equiv c \pmod{n}$ für ein $c \neq 1$. Mit Lemma 5 folgt, dass es für jeden Fermat-Lügner mindestens einen Fermat-Zeugen geben muss. \square

6.3 Der Miller-Rabin-Test

6.3.1 Eine verschärfte Version des Fermat-Tests

Bisher haben wir nur den kleinen Satz von FERMAT als notwendiges Kriterium dafür genutzt, dass eine Zahl n prim ist. Das Verfahren nach *Miller-Rabin*, ein weiterer probabilistischer Primzahltest, stützt sich zusätzlich auf folgendes Lemma.

Lemma 6. *Es sei p eine Primzahl. Ist a eine ganze Zahl mit $a^2 \equiv 1 \pmod{p}$ und $a \not\equiv 1 \pmod{p}$, so ist $a \equiv -1 \pmod{p}$.*

Beweis. Es gilt $p \mid (a^2 - 1) = (a - 1)(a + 1)$; nach Voraussetzung ist $a \not\equiv 1 \pmod{p}$, also $p \nmid a - 1$, mit Lemma 3 gilt $p \mid a + 1$, das heißt $a \equiv -1 \pmod{p}$. \square

Ist folglich $a^2 \equiv 1 \pmod{n}$, aber $a \not\equiv \pm 1 \pmod{n}$, so haben wir gezeigt, dass n zusammengesetzt ist. Diese Beobachtung rechtfertigt folgende Definition.

Definition 9. Es sei $n - 1 = 2^s \cdot t > 0$, wobei t ungerade ist. Eine ganze Zahl a heißt dann *Miller-Rabin-Zeuge* für die Zusammengesetztheit von n , wenn ein r mit $0 \leq r < s$ existiert, sodass $a^{2^r t} \not\equiv \pm 1 \pmod{n}$ und $(a^{2^r t})^2 = a^{2^{r+1} t} \equiv 1 \pmod{n}$, oder a ein Fermat-Zeuge für die Zusammengesetztheit von n ist.

Am Beispiel der CARMICHAEL-Zahl 561 mit $a = 61$ soll die Funktionsweise des Tests demonstriert werden (siehe Tab. 2). Mit $67^2 \equiv 1 \pmod{561}$ aber $67 \not\equiv \pm 1 \pmod{561}$ ist gezeigt, dass 561 zusammengesetzt ist. Die Potenzen von a sind jeweils noch in der Darstellung des chinesischen Restsatzes angegeben, wir wollen nämlich später untersuchen, wann eine Zahl a ein *Miller-Rabin-Lügner* ist, das heißt wann sie ein Fermat-Lügner ist und trotz der Zusammengesetztheit von n die Kongruenzen $a^{2^{r+1} t} \equiv 1 \pmod{n}$ und $a^{2^r t} \not\equiv \pm 1 \pmod{n}$ für kein $0 \leq r < s$ gelten. Dies ist genau dann der Fall, wenn alle Komponenten der Darstellung im chinesischen Restsatz von einer Zeile auf die nächste von -1 nach 1 wechseln.

Zahl	mod 3	mod 11	mod 17	mod 561
a	1	6	10	61
a^{35}	1	-1	14	439
$(a^{35})^2$	1	1	9	289
$(a^{35})^4$	1	1	13	166
$(a^{35})^8$	1	1	-1	67
$(a^{35})^{16}$	1	1	1	1

Tab. 2: Beispiel des verschärften Fermat-Tests mit $n = 561$ und $a = 61$

Der genaue Ablauf des Fermat-Tests ist in Abb. 9 dargestellt; wir werden nun beweisen, dass dieses Verfahren keine Primzahlen als zusammengesetzt erkennt.

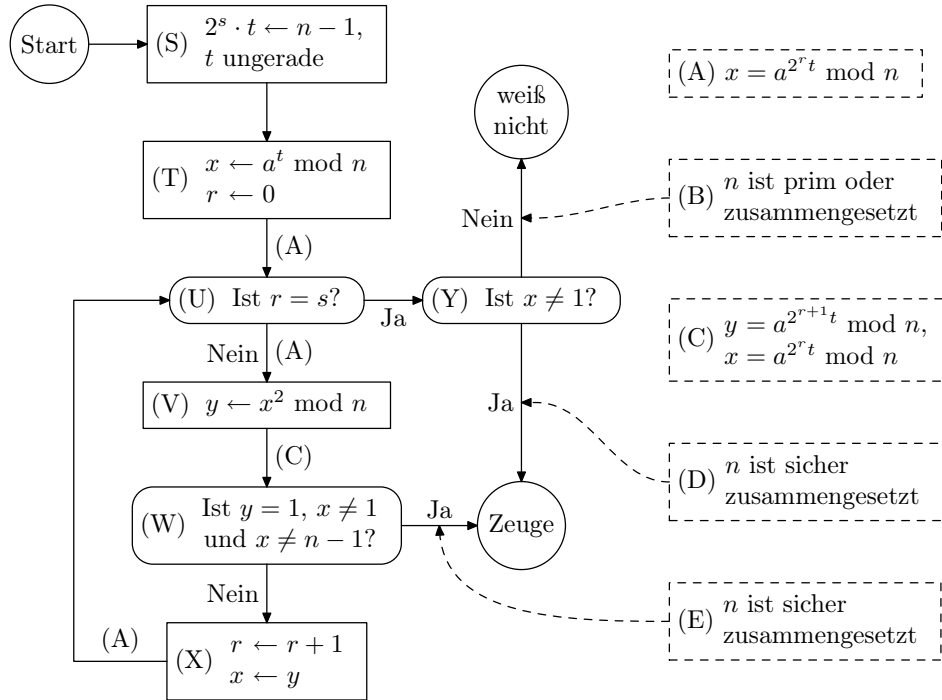


Abb. 9: Flussdiagramm und Invarianten des verschärften Fermat-Tests

Satz 19. *Klassifiziert der verschärfte Fermat-Test eine Basis a mit $0 < a < n$ als Zeugen für die Zusammengesetztheit einer ganzen Zahl $n > 1$, dann ist n zusammengesetzt.*

Beweis. Nach Schritt (T) gilt (A) wegen $r = 0$ trivialerweise und Schritt (U) ändert nichts an der Gültigkeit von (A). Wurde Schritt (V) ausgeführt, so liefert (A) den Zusammenhang $x = a^{2^r t} \bmod n$, mithin gilt $y = x^2 \bmod n = (a^{2^r t})^2 \bmod n = a^{2^{r+1} t} \bmod n$; (C) ist also bewiesen. Die Korrektheit von (A) nach (X) folgt durch die Substitution $r \leftarrow r + 1$ und $x \leftarrow y$, eingesetzt in Gleichung $y = a^{2^{r+1} t} \bmod n$ aus (C).

Wir beweisen nun die Korrektheit von (D) nach (Y). Die Invariante (A), die vor (U) gilt, liefert $x = a^{2^r t} \bmod n$, mit $r = s$ gilt $x = a^{n-1} \bmod n$. Ist also $x \neq 1$, so gilt $a^{n-1} \not\equiv 1 \pmod{n}$. Wenn n Primzahl war, so ist wegen $0 < a < n$ aber $a \perp n$, nach dem kleinen Satz von Fermat gilt $a^{n-1} \equiv 1 \pmod{n}$: Widerspruch zu $a^{n-1} \not\equiv 1 \pmod{n}$, folglich ist n zusammengesetzt, womit (D) gilt.

Genauso zeigen wir, dass nach (W) die Invariante (E) gilt. Erneut sei n zwecks eines Widerspruchs prim. (C) und (W) liefern $1 \equiv a^{2^{r+1}t} = (a^{2^r t})^2 \pmod{n}$ und $a^{2^r t} \not\equiv \pm 1 \pmod{n}$; nach Lemma 6 kann n nicht prim sein, (E) ist bewiesen. \square

Für den Fermat-Test der zusammengesetzten Zahl n wurde die Existenz von mindestens $(n-1)/2$ Zeugen in Satz 20 bereits bewiesen, wenn n keine CARMICHAEL-Zahl ist (was bei unserem „verschärften“ Fermat-Test natürlich weiterhin gilt). Wie wir aber nun zeigen, gilt die Abschätzung bei der Suche nach einem Miller-Rabin-Zeugen sogar für CARMICHAEL-Zahlen n , womit Miller-Rabin die Schwachstelle des Fermat-Tests nicht aufweist.

Satz 20. *Ist $n > 2$ eine CARMICHAEL-Zahl, so gibt es mindestens $(n-1)/2$ Miller-Rabin-Zeugen a mit $1 \leq a < n$, dass n zusammengesetzt ist.*

Beweis. Wir schreiben $n-1 = 2^s \cdot t$, wobei t ungerade ist und zerlegen die Menge $\{1, 2, \dots, n-1\}$ in disjunkte Teilmengen $A, B, C_1, C_2, \dots, C_s$, sodass

- $a \in A$, falls $a \not\perp n$,
- $a \in B$, falls $a^t \equiv 1 \pmod{n}$ und
- $a \in C_j$ für $1 \leq j \leq s$, falls $a^{2^j t} \equiv 1 \pmod{n}$ und $a^{2^{j-1}t} \not\equiv 1 \pmod{n}$.

In A kann kein Lügner sein, da für jeden Lügner a der Zusammenhang $a^{n-1} \equiv 1 \pmod{n}$ gilt, woraus mit Lemma 4 auch $a \perp n$ folgt: Widerspruch zu $a \not\perp n$.

Nachfolgend werden wir beweisen, dass $|B| \leq \frac{n-1}{4}$ und dass sich in jedem C_j höchstens $|C_j|/3$ Lügner befinden; die Anzahl der Lügner in $\{1, 2, \dots, n-1\}$ ist damit höchstens

$$|B| + \frac{n-1-|B|}{3} = \frac{2|B|}{3} + \frac{n-1}{3} \leq \frac{2(n-1)}{3 \cdot 4} + \frac{4(n-1)}{3 \cdot 4} = \frac{n-1}{2}$$

Abschätzung für $|B|$. Nach Satz 17 ist $n = p_1 p_2 \dots p_l$ ein Produkt $l \geq 2$ verschiedener Primzahlen. Nach dem chinesischen Restsatz sei $a = (a_1, a_2, \dots, a_l)$ mit dem Modul $n = p_1 p_2 \dots p_l$. Mit Satz 13 ist $a \in B$ genau dann, wenn $a_k^t \equiv 1 \pmod{p_k}$ für $1 \leq k \leq l$.

Wir zeigen zuerst, dass $a_k^t \equiv 1 \pmod{p_k}$ höchstens für die Hälfte der Elemente $a_k \in \{1, 2, \dots, p_k-1\}$ gilt. Gibt es kein solches Element b , so sind wir fertig; andernfalls ist $(-b)^t = -b^t \equiv -1 \pmod{p_k}$, wegen t ungerade. Mit $-1 \equiv p_k-1 \perp p_k$, wegen Lemma 4 auch $(-b) \perp p_k$, existiert nach Lemma 5 für jedes $a_k^t \equiv 1 \pmod{p_k}$ mindestens ein $a_k^t \not\equiv 1 \pmod{p_k}$.

Sei also $(a_1, a_2, \dots, a_l) \in B$, womit für alle $1 \leq k \leq l$ die Kongruenz $a_k^t \equiv 1 \pmod{p_k}$ gilt. Es gibt maximal $\frac{p_1-1}{2} \cdot \frac{p_2-1}{2} \cdot \dots \cdot \frac{p_l-1}{2}$ solcher Paare, wegen $l \geq 2$ gilt dann

$$|B| \leq \frac{p_1-1}{2} \cdot \frac{p_2-1}{2} \cdot \dots \cdot \frac{p_l-1}{2} \leq \frac{n-1}{2^l} \leq \frac{n-1}{4}.$$

Abschätzung für $|C_j|$. Auch hier betrachten wir $(a_1, a_2, \dots, a_l) \in C_j$, sodass $a_k \in \{1, 2, \dots, p_k - 1\}$ und $a_k^{2^j t} \equiv 1 \pmod{p_k}$ für alle $1 \leq k \leq l$. Nach Lemma 6 folgt wegen p_k prim für jedes j aus $a_k^{2^j t} \equiv 1 \pmod{p_k}$ die Beziehung $a_k^{2^{j-1} t} \equiv \pm 1 \pmod{p_k}$. Wir definieren $\sigma_a = a^{2^{j-1} t} = (a_1^{2^{j-1} t}, a_2^{2^{j-1} t}, \dots, a_l^{2^{j-1} t})$ für $a \in C_j$. Nie ist $\sigma_a = (1, 1, \dots, 1)$, weil ansonsten $a \in C_h$ mit $h < j$. Ein Element a aus C_j ist genau dann ein Miller-Rabin-Lügner, wenn $\sigma_a = (-1, -1, \dots, -1)$, weil dann mit (14) gilt $a^{2^{j-1} t} \equiv -1 \pmod{n}$.

Wir wollen nun beweisen, dass C_j höchstens $|C_j|/3$ Miller-Rabin-Lügner enthält. Enthält C_j einen solchen Lügner (ansonsten müssen wir nichts beweisen), dann hat jede der Kongruenzen $b^{2^{j-1} t} \equiv -1 \pmod{p_k}$ für $1 \leq k \leq l$ eine Lösung b_k . Sei $w_k = (1, 1, \dots, b_k, \dots, 1)$, also $w_k \equiv b_k \pmod{p_k}$ und $w_k \equiv 1 \pmod{p_{k'}}$ für jedes $k' \neq k$. Unterscheiden sich σ_c und σ_d genau im Vorzeichen der k -ten Komponente, dann ist $c \cdot w_k = (c_1 \cdot w_1, \dots, c_k \cdot w_k, \dots, c_l \cdot w_l) = (c_1, \dots, c_k \cdot w_k, \dots, c_l)$ und damit

$$\sigma_{cw_k} = (\dots, c_k^{2^{j-1} t} \cdot w_k^{2^{j-1} t}, \dots) = (\dots, c_k^{2^{j-1} t} \cdot (-1), \dots) = \sigma_d.$$

Die eindeutige⁸ Abbildung $z \mapsto zw_k$ ordnet also jedem $c \in C_j$ mit der „Vorzeichenbelegung“ σ_c ein $d \in C_j$ mit Vorzeichenbelegung σ_d zu (und umgekehrt), womit jede der $2^l - 1$ möglichen Vorzeichenbelegungen eine Teilmenge von C_j festlegt, deren Mächtigkeit genau $|C_j|/(2^l - 1)$ beträgt. Wegen $l \geq 2$ hat insbesondere die Zeichenfolge der Miller-Rabin-Lügner $(-1, -1, \dots, -1)$ die Kardinalität $|C_j|/(2^l - 1) \leq |C_j|/3$. \square

6.3.2 Mehrere Zeugen sind noch zuverlässiger: Der Miller-Rabin-Test

Beim Fermat-Test von n wäre es wenig hilfreich gewesen, mehrere Basen a als Zeugen zu „befragen“ da CARMICHAEL-Zahlen nie erkannt werden. Der Test von n zur Basis a aus Abschnitt 6.3.1 kann aber für *jede* Zahl mit 50% Wahrscheinlichkeit die Zusammengesetztheit von n zeigen. In der Praxis wählt man zufällig k unabhängige Basen a ; die Irrtumswahrscheinlichkeit beträgt dann $1/2^k$. Knuth schreibt in Knuth (1998, S. 395)

„But if the algorithm reports [50] times in a row that n is “probably prime,” we can say that n is “almost surely prime.” For the probability is less than $[(1/2)^{50}]$ that such a [50]-times-in-a-row procedure gives the wrong information about its input. This is less than one chance in a quadrillion; [...] It’s much more likely that our computer has dropped a bit in its calculation, due to hardware malfunctions or cosmic radiations, than that [the algorithm] has repeatedly guessed wrong!“

Die mehrmalige Ausführung des verschärften Fermat-Tests nennt man *Miller-Rabin-Test*.

7 Die Sicherheit der RSA-Verschlüsselung

Einen Nachweis bin ich dem Leser noch schuldig geblieben. In der Einleitung haben wir zwar als Grundannahme aufgestellt, dass es schwer ist, eine große Zahl in ihre Primfaktoren zu zerlegen, aber die Sicherheit der RSA-Verschlüsselung, bzw. die Tatsache, dass f_O eine Einwegfunktion ist, folgt daraus alleine noch nicht. Wenn man die Primfaktorzerlegung

⁸wegen $w_k \not\equiv 0 \pmod{p_k}$ ist $w_k \perp p_k$, womit die Abbildung $z_k \mapsto z_k w_k$ nach Satz 11 umkehrbar ist.

von $n = pq$ hat, kann man d berechnen, der private Schlüssel ist geknackt. Wir müssen nun beweisen, dass man auch umgekehrt aus d die Primfaktorzerlegung von n effizient berechnen kann, womit beide Probleme äquivalent sind.

Lemma 7. *Sei $n = pq$ ein Produkt aus zwei Primzahlen $p > 2$ und $q > 2$. Gibt es ein a und ein m mit $a^{2m} \equiv 1 \pmod{n}$ und $a^m \equiv \pm 1 \pmod{p}$ sowie $a^m \equiv \mp 1 \pmod{q}$, das heißt a^m hat verschiedene Vorzeichen modulo p und q , so ist a ein Spion der Primfaktorzerlegung von n , mit a und m kann also p und q berechnet werden.*

Beweis. Im Falle $a^m \equiv +1 \pmod{p}$ und $a^m \equiv -1 \not\equiv 1 \pmod{q}$ gilt $p \mid a^m - 1$ und $q \nmid a^m - 1$, womit $\text{ggT}(a^m - 1, n) = \text{ggT}(a^m - 1, pq) = p$; andernfalls ist $\text{ggT}(a^m - 1, pq) = q$. Den ggT können wir mit dem Verfahren aus Abschnitt 4.1 effizient berechnen. \square

Um folgenden Hilfssatz zu zeigen, gehen wir ähnlich wie beim Beweis von Satz 20 vor.

Lemma 8. *In der Menge $\{1, 2, \dots, n - 1\}$ gibt es mindestens $(n - 1)/2$ Spione a der Primfaktorzerlegung von $n = pq$.*

Beweis. Es sei $ed - 1 = 2^s \cdot t$ mit t ungerade, sodass nach Satz 15 gilt $a^{2^s t} \equiv 1 \pmod{n}$. Der Beweis von Satz 20 ist unabhängig vom konkreten Wert des Exponenten $2^s t$ und $n = pq$ erfüllt die notwendige Bedingung von Satz 17 für eine CARMICHAEL-Zahl. Deswegen gibt es nach Satz 20 mindestens $(n - 1)/2$ „Miller-Rabin-Zeugen“ a , sodass wegen $a^{2^s t} \equiv 1 \pmod{n}$ nach Definition 9 ein r mit $a^{2^r t} \not\equiv \pm 1 \pmod{n}$ und $a^{2^{r+1} t} \equiv 1 \pmod{n}$ für $0 \leq r < s$ existiert. Modulo pq betrachtet gilt $a^{2^r t} = (\pm 1, \mp 1)$, womit a mit $m = 2^r \cdot t$ ein Spion ist. \square

Satz 21. *Hat man den öffentlichen RSA-Schlüssel (e, n) und den geheimen Schlüssel (d, n) , so kann man n „leicht“ faktorisieren.*

Beweis. Man wähle ein zufälliges $a \in \{1, 2, \dots, n - 1\}$ und berechne s und t , sodass $ed - 1 = 2^s \cdot t$, wobei t ungerade ist. Falls $a^t \not\equiv 1 \pmod{n}$ ist, bestimmt man durch fortgesetzte Quadrierung das r mit $0 \leq r < s$, sodass $a^{2^{r+1} t} \equiv 1 \pmod{n}$ und $a^{2^r t} \not\equiv 1 \pmod{n}$. Ist a mit $m = 2^r \cdot t$ nach Lemma 7 ein Spion, so ist man fertig; ansonsten wählt man das nächste a . Nach Lemma 8 ist nach u unabhängigen Tests die Wahrscheinlichkeit, einen Spion gefunden zu haben $1 - 1/2^u$ (vgl. Abschnitt 6.3.2). \square

Damit ist das Problem „Modul n faktorisieren“ genauso „schwer“ wie das Problem „den geheimen Schlüssel (d, n) bestimmen“. Das bedeutet aber keineswegs, dass RSA sicher ist. Erstens wird in (ferner?) Zukunft die Grundannahme „Faktorisieren ist schwer“ nicht mehr stimmen, da laut Wikipedia (2008d) Peter Shor einen Algorithmus gefunden hat, mit dem auf einem Quantencomputer eine Zahl effizient faktorisiert werden kann. Zweitens ist nach Buchmann (2008, Abschnitt 9.3.5) die Frage, ob eine RSA-verschlüsselte Nachricht auch ohne Kenntnis des geheimen Schlüssels gelesen werden kann, ungeklärt.

A Implementierung der Algorithmen in Common Lisp

A.1 Zahlentheoretische Algorithmen

Als erstes betrachten wir den erweiterten Algorithmus von Euklid, der nach dem Flussdiagramm von Abb. 5 umgesetzt ist.

22a \langle Der erweiterte Algorithmus von Euklid 22a $\rangle \equiv$ (22c)

```
(defun extended-euclid (x y)
  "Calculate  $d$ ,  $m$ ,  $n$  such that  $d = xm + yn = \gcd(x, y)$ ."
  (let ((u 1) (v 0) (m 0) (n 1))
    (loop for q = (floor x y) and r = (mod x y)
          when (= r 0) return (values y m n) do
            (psetf u m m (- u (* q m)))
            (psetf v n n (- v (* q n)))
            (shiftf x y r))))
```

Nun folgt eine Implementierung der modularen Exponentiation aus Abschnitt 4.2, welche die Grundlage für viele andere Funktionen bildet (siehe Abb. 6).

22b \langle Die modulare Exponentiation 22b $\rangle \equiv$ (23d)

```
(defun modular-power (a b m)
  "Calculate  $a^b \bmod m$  by repeated squaring."
  (loop for k from (- (integer-length b) 1) downto 0
        with d = 1 finally (return d) do
          (setf d (mod (* d d) m))
          (when (logbitp k b)
            (setf d (mod (* d a) m)))))
```

Auf der Basis des euklidischen Algorithmus können wir wie in Abschnitt 4.3.1 hergeleitet für $a \perp m$ das multiplikative Inverse zu a finden.

22c \langle Berechnung des multiplikativen Inversen 22c $\rangle \equiv$ (22d)

\langle Der erweiterte Algorithmus von Euklid 22a \rangle

```
(defun find-inverse (a m)
  "Assuming  $a \perp m$ , find  $x$  such that  $ax \equiv 1 \pmod m$ ."
  (multiple-value-bind (d a b)
    (extended-euclid a m) (declare (ignore d b))
    (mod a m)))
```

A.2 Der RSA-Algorithmus

Nun betrachten wir die Umsetzung der Funktion, welche die Schlüssel der Bitlänge `2·bit-length` generiert. Sie nutzt die Hilfsfunktion `generate-prime`, die wir später noch definieren werden. Die Schlüsselerzeugung wurde in Abschnitt 5.2 untersucht.

22d \langle Erzeugen der RSA-Schlüssel 22d $\rangle \equiv$ (25)

\langle Berechnung des multiplikativen Inversen 22c \rangle

\langle Generiere (wahrscheinlich) eine Primzahl mit `bit-length Bits` 24c \rangle

```
(defun generate-keys (bit-length)
  "Generate a public and a private key of bit length $2 bit-length$."
  (let* ((p (generate-prime bit-length))
        (q (generate-prime bit-length))
        (n (* p q)) (k (* (- p 1) (- q 1)))
        (e (+ (expt 2 16) 1)) (d (find-inverse e k)))
    (values (cons e n) (cons d n))))
```

Mit den so generierten Schlüsseln kann man nun mithilfe der Funktion `encode` eine Nachricht ver-/entschlüsseln, je nachdem ob man den öffentlichen oder den privaten Schlüssel als Argument übergibt. Die Funktion `encrypt` stimmt mit der Funktion $f_S(m)$ aus Abschnitt 5.2 überein.

23a $\langle \text{Ver-/Entschlüsseln der Nachricht message 23a} \rangle \equiv$ (25)

```
(defun encrypt (message key)
  "Encrypt (or decrypt) $message$ with $key$."
  (modular-power message (car key) (cdr key)))
```

A.3 Die Suche nach Primzahlen

Als letztes müssen wir die Funktion `generate-prime` umsetzen; intern wird der Miller-Rabin-Test mit $k = 50$ genutzt, zuvor wird jedoch wie in Abschnitt 6.1 beschrieben das Verfahren zur Probedivision angewendet. Die Primzahlen dafür werden mit dem Sieb des Eratosthenes generiert.

23b $\langle \text{Sieb des Eratosthenes 23b} \rangle \equiv$ (23c)

```
(defun eratosthenes-sieve (num)
  "Return a list of primes $p$ with $p \leq$ num$."
  (let ((table (make-array (1+ num) :element-type 'bit
                           :initial-element 1)))
    (loop for n from 2 upto num do
      (loop for k = (+ n n) then (+ k n) while (<= k num) do
        (setf (aref table k) 0)))
    (loop for j from 2 upto num when (= (aref table j) 1) collect j)))
```

Die Probedivision wird mit den Primzahlen bis $2^{21} \approx 2,1 \cdot 10^6$ durchgeführt, womit die meisten zusammengesetzten Zahlen schon aussortiert sind, bevor sie mit Miller-Rabin getestet werden.

23c $\langle \text{Primzahl-Liste bis } 2^{21} \text{ 23c} \rangle \equiv$ (24c)

$\langle \text{Sieb des Eratosthenes 23b} \rangle$

```
(defparameter *prime-list* (eratosthenes-sieve (expt 2 21)))
```

Wir kommen nun zum Miller-Rabin-Test. Als erstes implementieren wir den verschärften Fermat-Test aus Abschnitt 6.3.1.

23d $\langle \text{Ist a ein Zeuge für die Zusammengesetztheit von n? 23d} \rangle \equiv$ (24a)

$\langle \text{Die modulare Exponentiation 22b} \rangle$

```
(defun witness (a n)
```

```

"Is $a$ a witness of the compositeness of $n$?"
(let* ((s (loop for k from 0 until (logbitp k (- n 1))
                count t into low-bits finally (return low-bits)))
      (d (ash (- n 1) (- s)))
      (x (modular-power a d n)) (y 0))
  (dotimes (k s)
    (setf y (mod (* x x) n))
    (when (and (= y 1) (not (= x 1)) (not (= x (- n 1))))
      (return-from witness t))
    (setf x y))
  (/= x 1)))

```

Diese Funktion `witness` nutzt der Miller-Rabin-Test wie in Abschnitt 6.3.2 angedeutet, um die Zahl n genau k -mal darauf zu testen, ob sie eine Primzahl ist.

24a \langle Der Miller-Rabin-Test 24a $\rangle \equiv$ (24c)
 \langle Ist a ein Zeuge für die Zusammengesetztheit von n ? 23d \rangle

```

(defun miller-rabin (n k)
  "Test $n$ for primeness with the Miller-Rabin primality test."
  (dotimes (j k)
    (when (witness (+ (random (- n 1)) 1) n)
      (return-from miller-rabin nil)))
  t)

```

Um große Primzahlen zu generieren, müssen wir erst große Zufallszahlen erzeugen, die dann darauf getestet werden, ob sie Primzahlen sind. Die Funktion `random-odd-number` nutzt den Pseudo-Zufallsgenerator von Common-Lisp, um lauter einzelne Zufallsbits zu erzeugen, die dann zu einer ungeraden Zufallszahl mit `bit-length` Bits zusammengebaut werden.

24b \langle Erzeuge große, ungerade Zufallszahlen 24b $\rangle \equiv$ (24c)

```

(defun random-odd-number (bit-length)
  "Generate a random odd number with exactly $bit-length > 1$ bits."
  (+ (loop for k from 1 below (- bit-length 1)
          sum (ash (random 2) k))
     1 (ash 1 (- bit-length 1))))

```

Die Funktion `generate-prime` nutzt die Probedivision und Miller-Rabin, um die Primzahlen zu generieren. Das Prädikat `pred` testet, ob x ein echter Teiler von n ist. Die Anzahl der Miller-Rabin Iterationen wurde auf $k = 50$ festgesetzt (siehe Abschnitt 6.3.2).

24c \langle Generiere (wahrscheinlich) eine Primzahl mit `bit-length` Bits 24c $\rangle \equiv$ (22d)
 \langle Primzahl-Liste bis 2^{21} 23c \rangle
 \langle Der Miller-Rabin-Test 24a \rangle
 \langle Erzeuge große, ungerade Zufallszahlen 24b \rangle

```

(defun generate-prime (bit-length)
  "Generate a prime with $bit-length$ bits."
  (flet ((pred (x n) (and (/= n x) (= (mod n x) 0))))

```

```
(loop for n = (random-odd-number bit-length)
      when (and (notany #'(lambda (x) (pred x n)) *prime-list*)
                (miller-rabin n 50)) return n)))
```

A.4 Praktische Anwendung des Verfahrens

Wenn man das Verfahren praktisch nutzen will, ist noch einiges an Arbeit nötig. Ich habe nur eine kleine Testfunktion geschrieben, anhand der unser Satz 15 anhand einiger Beispiele nachvollzogen werden kann.

```
25  ⟨ * 25 ⟩ ≡
    ⟨ Erzeugen der RSA-Schlüssel 22d ⟩
    ⟨ Ver-/Entschlüsseln der Nachricht message 23a ⟩
    (defun test (message)
      (multiple-value-bind (public-key private-key) (generate-keys 1024)
        (encrypt (encrypt message public-key) private-key))))
```

Die Funktion verschlüsselt den Klartext erst mit dem öffentlichen 2048-Bit Schlüssel und entschlüsselt dann den Geheimtext wieder mit dem privaten Schlüssel. Man kann mit Norman Ramseys `noweb`, dem Lisp-Compiler `sbcl` und `main.noweb`, dem \TeX -Source dieser Facharbeit, das Programm folgendermaßen testen.

```
pcmoritz@earth:~/facharbeit$ notangle main.noweb > rsa.lisp
pcmoritz@earth:~/facharbeit$ sbcl
* (load "rsa.lisp")
* (test 561)
561
```

In der Praxis wird man für längere Nachrichten nicht direkt den RSA-Algorithmus nutzen, sondern eines der symmetrischen Verfahren, die im Allgemeinen schneller sind als ihre asymmetrischen Pendanten. RSA wird man nur nutzen, um das Problem aus der Einleitung, den Schlüsselaustausch, zu lösen.

Eine weitere Anwendung des von RSA ist das *Signieren* von Daten. Verschlüsselt Alice diese mit ihrem privaten Schlüssel, so kann jeder mit dem öffentlichen Schlüssel sicherstellen, dass sie von Alice stammen. Auch hier wird man in der Praxis nicht die Nachricht selbst, sondern einen *Hash-Wert* der Nachricht signieren.

B Komplexität der Algorithmen

Wie wir am Faktorisierungsproblem gesehen haben, ist nicht nur die Korrektheit eines Algorithmus wichtig (die ich hoffentlich für alle verwendete Algorithmen überzeugend genug gezeigt habe), sondern auch dessen *Effizienz*. Um diese zu analysieren nutzt man das *Landau-Symbol* \mathcal{O} ; ich gehe im folgenden Abschnitt davon aus, dass dessen Definition und Eigenschaften bekannt sind, ansonsten sollte der Leser Graham u. a. (1997, Abschnitt 9.2) zu Rate ziehen.

Wir sagen, die *Laufzeit* eines Algorithmus mit der (vorher zu spezifizierenden) Eingabegröße n ist $\mathcal{O}(g(n))$, wenn es ein $f(n) \in \mathcal{O}(g(n))$ gibt, sodass der Wert $f(n)$ größer oder gleich dem Rechenaufwand zur Verarbeitung einer Eingabe der Größe n ist. Man nennt einen Algorithmus „effizient“, wenn er die Laufzeit $\mathcal{O}(f(n))$ hat, wobei $f(n)$ ein Polynom in n ist.

Zahlentheoretische Algorithmen

Für die Analyse der grundlegenden arithmetischen Operationen nehmen wir an, dass alle betrachteten Zahlen a die maximale Bitlänge n besitzen, also $a < 2^{n+1}$ ist.

Lemma 9. *Die Addition zweier Zahlen a und b ist in $\mathcal{O}(n)$ möglich, das Produkt von a und b kann man in $\mathcal{O}(n^2)$ berechnen. Der Quotient und der Rest von a durch b sind jeweils in $\mathcal{O}(n^2)$ berechenbar.*

Beweis. Alle Abschätzungen trivial aus den Rechenmethoden aus der Grundschule. \square

Für die Analyse der modularen Exponentiation sei n die Bitlänge des Moduls m und es gelte $a \leq m$ und $b \leq m$. Dann gilt folgende Abschätzung.

Satz 22. *Die modulare Exponentiation hat eine Laufzeit von $\mathcal{O}(n^3)$.*

Beweis. Zuerst berechnen wir die Zeit, die man zum Quadrieren modulo m braucht. Die Zahl $a^2 = a \cdot a$ ist nach Lemma 9 in $\mathcal{O}(n^2)$ berechenbar. Das Reduzieren modulo m benötigt $\mathcal{O}(n^2)$, insgesamt braucht Quadrieren modulo m die Laufzeit $\mathcal{O}(n^2 + n^2) = \mathcal{O}(2n^2) = \mathcal{O}(n^2)$. Da wir n -mal Quadrieren ist dafür die Laufzeit $\mathcal{O}(n^2) \cdot n = \mathcal{O}(n^3)$ nötig. Das Testen, ob ein Bit gesetzt ist, ist in konstanter Laufzeit $\mathcal{O}(1)$ möglich. Wir müssen n mal testen und maximal n mal die Quadrate aufaddieren; insgesamt ergibt sich die Laufzeit $n \cdot \mathcal{O}(1) + n \cdot \mathcal{O}(n) + \mathcal{O}(n^3) = \mathcal{O}(n^3)$. \square

Beim erweiterten euklidischen Algorithmus soll n die Bitlänge der Zahl a mit $a > b$ sein.

Satz 23. *Mit $a > b \geq 1$ hat der erweiterte euklidische Algorithmus die Laufzeit $\mathcal{O}(n^3)$.*

Beweis. Es sei $m \geq 1$ definiert durch $\text{ggT}(a, b) = r_m$, das heißt Schritt (T) wird m -mal ausgeführt; durch Induktion über m beweisen wir, dass $a \geq F_{m+2}$ und $b \geq F_{m+1}$, wobei F_k die k -te Fibonaccizahl ist. Für $m = 1$ ist offensichtlich $b \geq 1 = F_2$ und mit $a > b$ auch $a \geq 2 = F_3$. War $\text{ggT}(a, b) = r_m$, dann gilt für $\text{ggT}(a', b') = r_{m+1}$ nach Induktionsvoraussetzung $b' = a \geq F_{m+3}$ und wegen $a' = q' \cdot a + b$ und $q' \neq 0$ (weil $a' > b'$) ist $a' \geq F_{m+2} + F_{m+1} = F_{m+3}$. Wegen⁹

$$F_k = \frac{1}{\sqrt{5}} (\phi^k - \phi^{-k}), \quad \text{wobei } \phi = \frac{1 + \sqrt{5}}{2},$$

ist für große k dann $F_k \approx \phi^k / \sqrt{5}$. Insgesamt bekommt man mit $a \geq F_{m+2} \approx \phi^{m+2} / \sqrt{5}$ also $\log_\phi(a) \geq m + 2 - \log_\phi(\sqrt{5})$, womit $m = \mathcal{O}(\log(a)) = \mathcal{O}(n)$. Der Algorithmus führt also $\mathcal{O}(n)$ -mal verschiedene Operationen der maximalen Komplexität $\mathcal{O}(n^2)$ aus, womit er die Laufzeit $\mathcal{O}(n^3)$ besitzt. \square

⁹siehe Moritz (2006, Abschnitt 6.2.1)

Korollar 1. Auch das Verfahren zum Bestimmen des multiplikativen Inversen von a modulo m ist in $\mathcal{O}(n^3)$ berechenbar, wobei n die Bitlänge des Moduls $m > a$ ist.

Beweis. Erweiterter Algorithmus von Euklid in $\mathcal{O}(n^3)$ und Rest modulo m in $\mathcal{O}(n^2)$. \square

Die Suche nach Primzahlen

Die Primzahlliste ist offensichtlich in konstanter Laufzeit $\mathcal{O}(1)$ zu berechnen, da 2^{21} konstant ist. Wir betrachten die Laufzeit des verschärften Fermat-Test einer n -Bit Zahl.

Lemma 10. Der verschärfte Fermat-Test einer n -Bit Zahl m läuft in $\mathcal{O}(n^3)$ ab.

Beweis. Die Zahlen s und d , sodass $m - 1 = 2^s \cdot d$ sind in $\mathcal{O}(n)$ berechenbar, da nur Bittests und -shifts nötig sind. Man kann $a^d \bmod m$ in $\mathcal{O}(n^3)$ berechnen, da $d < m \leq a$; außerdem sind s Quadrierungen in $s \cdot \mathcal{O}(n^2) = \mathcal{O}(n^3)$ möglich. Das Testen einer Zahl braucht $\mathcal{O}(n)$; insgesamt erhalten wir $\mathcal{O}(n^3)$ als obere Grenze der Laufzeit. \square

Wenn wir davon ausgehen, dass n -Bit Zufallszahlen in $\mathcal{O}(n)$ erzeugbar sind, läuft der Miller-Rabin-Test in $k \cdot \mathcal{O}(n^3) = \mathcal{O}(n^3)$ ab. Die wichtige Frage, wie viel Laufzeit **generate-prime** braucht, kann ich im Rahmen dieser Arbeit leider nicht endgültig klären, da hierzu der „Primzahlsatz“ benötigt würde. Die problematische Größe X ist, wieviele Zahlen getestet werden müssen, bis eine Primzahl gefunden wird. In Tab. 3 sind experimentelle Werte für relevante Bitlängen aufgeführt, die bestätigen sollen, dass X nicht allzu groß wird.

Bitlänge n	\bar{X}	Abweichung
512	182,46	± 814
1024	360,07	± 1514
2048	697,62	± 3274

Tab. 3: Durchschnittswert \bar{X} von X und maximale lineare Abweichung vom Durchschnitt bei je 100 Tests für die Bitlängen $n \in \{512, 1024, 2048\}$

Insgesamt ist zum Erzeugen einer n -Bit Primzahl die Laufzeit $\mathcal{O}(X \cdot n^3)$ nötig, da der Miller-Rabin-Test maximal X -mal ausgeführt werden muss (in der Praxis werden die meisten zusammengesetzten Zahlen vorher durch die Probedivision „ausgesiebt“).

Der RSA-Algorithmus

Die Schlüsselerzeugung eines $2n$ -Bit Schlüsselpaares ist in $\mathcal{O}(2X \cdot n^3 + n^2 + n^3 + 1) = \mathcal{O}(X \cdot n^3)$ möglich. Verschlüsseln kann man mit der modularen Exponentiation in $\mathcal{O}(n^3)$.

C Nachweis der übernommenen Beweise

Die meisten Beweise, die in dieser Arbeit geführt werden, wurden nicht durch den Autor erarbeitet. Daher sollen in diesem Absatz die Quellen aller nicht-trivialen Beweise angegeben werden, die den Arbeiten anderer entnommen wurden.

Der Beweis zu Satz 2 entstand durch eine Kombination von Matzat (1992, Abschnitt 1.2) mit Wikipedia (2008c, Abschnitt “Proof”). Der Beweis zu Satz 3 ist durch Meinel und Mundhenk (2006, Lemma 13.2) inspiriert, auch wenn er anders geführt ist. Der Beweis zu Lemma 1 wurde fast unverändert aus Wikipedia (2008a, Abschnitt “Proof”) übernommen (noch eleganter geht leider nicht). Der Beweis zu Satz 4 stammt aus Bamler u. a. (2005, Aufgabe d aus Abschnitt 4.2). Der erste Teil des Beweises von Satz 5 ist aus Knuth (1997, Übung 5 aus Abschnitt 1.2.1); der zweite Teil wird in Hoffmann (2003, Abschnitt 1.6) bewiesen.

Der Beweis zu Satz 9 stammt aus Knuth (1997, Abschnitt 1.2.1), der zu Satz 10 wurde mit Hilfe von Cormen u. a. (2007, Abschnitt 31.6) erarbeitet. Der Beweis zu Satz 11 ist aus Buchmann (2008, Abschnitt 3.6). Die Idee des Beweises von Satz 12 stammt aus Wikipedia (2008b) und Knuth (1998, Abschnitt 4.3.2); daraus stammt auch der Beweis von Satz 13. Der Beweis von Satz 14 wurden durch Bamler u. a. (2005, Abschnitt 4.7.2) inspiriert.

Das Verfahren aus Abschnitt 5.2 wurde aus Meinel und Mundhenk (2006, Abschnitt 13.7) und Cormen u. a. (2007, Abschnitt 31.7) kombiniert. Der grobe Ablauf des Beweises von Satz 15 entstammt Cormen u. a. (2007, Abschnitt 31.7). Der Beweis, dass 561 eine CARMICHAEL-ZAHL ist (siehe Abschnitt 6.2), wurde Schott (2004, Abschnitt 4.2.2) entnommen. Der Beweis von Satz 18 ist, inklusive der Hilfssätze, aus Kleinberg (2008, Korollar 9), wenn auch mit vielen Anpassungen. Der Beweis von Lemma 6 wurde Ruppert (2001/2002, Abschnitt 3.3) entnommen. Der Beweis von Satz 20 war am kompliziertesten: Nur eine Kombination der Tricks von Kleinberg (2008, Satz 15), Buchmann (2008, Satz 8.4.3) und Cormen u. a. (2007, Satz 31.38) sowie die Abschwächung der Schranke von $1/4$ auf $1/2$ haben den Beweis in dieser Form möglich gemacht. Damit wurde auch der Beweis zu Satz 21 samt Hilfssätze aus Buchmann (2008, Satz 9.3.8) und Stein (2007, Algorithmus 3.3.5) einfacher.

Der Beweis von Satz 23 stammt aus Cormen u. a. (2007, Abschnitt 31.8); der Beweis für die Formel von F_k ist aus Moritz (2006, Abschnitt 6.2.1).

Literatur

- [Bamler u. a. 2005] BAMLER, Richard ; REIHER, Christian u. a.: *Ein-Blick in die Mathematik*. Köln : Aulis Verlag Deubner, 2005
- [Bartholomé u. a. 2008] BARTHOLOMÉ, Andreas ; RUNG, Josef ; KERN, Hans: *Zahlentheorie für Einsteiger*. 6., überarbeitete und erweiterte Auflage. Wiesbaden : Vieweg und Teubner, 2008
- [Buchmann 2008] BUCHMANN: *Einführung in die Kryptographie*. 4., erweiterte Auflage. Berlin und Heidelberg : Springer Verlag, 2008
- [Cormen u. a. 2007] CORMEN, Thomas H. ; LEISERSON, Charles E. ; RIVEST, Ronald L. ; STEIN, Clifford: *Introduction to Algorithms*. 2. Auflage. Cambridge, Massachusetts : The MIT Press, 2007
- [Gallenbacher 2007] GALLENBACHER, Jens: *Abenteuer Informatik: IT zum Anfassen von Routenplaner bis Online-Banking*. München : Elsevier, 2007
- [Graham 1996] GRAHAM, Paul: *ANSI Common Lisp*. New York : Prentice Hall, 1996
- [Graham u. a. 1997] GRAHAM, Ronald L. ; KNUTH, Donald E. ; PATASHNIK, Oren: *Concrete mathematics: a foundation for computer science*. 2. Auflage. Addison-Wesley, 1997
- [Hoffmann 2003] HOFFMANN, Norbert: *Kryptographie*. Vorlesung am Mathematischen Institut Georg-August-Universität Göttingen. 2003. – URL www.uni-math.gwdg.de/kersten/kryptographie.ps
- [Kleinberg 2008] KLEINBERG, Bobby: The Miller-Rabin Randomized Primality Test. In: *Cornell University Lecture Notes* (2008). – URL <http://www.cs.cornell.edu/courses/cs482/2008sp/handouts/mrpt.pdf>
- [Knuth 1997] KNUTH, Donald E.: *The Art of Computer Programming*. Bd. 1: *Fundamental Algorithms*. 3. Auflage. Reading, Massachusetts : Addison-Wesley, 1997
- [Knuth 1998] KNUTH, Donald E.: *The Art of Computer Programming*. Bd. 2: *Seminumerical Algorithms*. 3. Auflage. Reading, Massachusetts : Addison-Wesley, 1998
- [Matzat 1992] MATZAT, B. H.: *Elementare Zahlentheorie*. Vorlesung an der Universität Heidelberg. 1992. – URL mathphys.fsk.uni-heidelberg.de/skripte/Files/Mathe/Zahlentheorie/Matzat/Zahlentheorie.ps
- [Meinel und Mundhenk 2006] MEINEL, Christoph ; MUNDHENK, Martin: *Mathematische Grundlagen der Informatik*. 3. Auflage. B. G. Teubner Verlag, 2006
- [Meiringer 2008] MEIRINGER, Markus: Sichere Kommunikation über unsichere Leitungen? In: *Computeralgebra Rundbrief* (2008), April, S. 48–52. – Sonderheft zum Jahr der Mathematik

- [Moritz 2006] MORITZ, Philipp: *Der goldene Schnitt*. Juli 2006
- [Pracht und Heidenreich 1978] PRACHT, Egon ; HEIDENREICH, Karl: *Elementare Zahlentheorie*. Paderborn : Schöningh, 1978
- [Ribenboim 2006] RIBENBOIM, Paulo: *Die Welt der Primzahlen*. Berlin und Heidelberg : Springer Verlag, 2006
- [Ruppert 2001/2002] RUPPERT, Wolfgang M.: *Algorithmische Zahlentheorie*. Script zur Vorlesung an der Universität Erlangen. 2001/2002. – URL <http://www.mi.uni-erlangen.de/~ruppert/skripten/algo.ps.gz>
- [Schott 2004] SCHOTT, Katharina: *Primzahlerkennung und Primzahltests*. Proseminarausarbeitung. 2004. – URL <http://mspcdip.mathematik.uni-karlsruhe.de/Blaetter/proseminar/pdf/Primzahltests.pdf>
- [Schwenk 2005] SCHWENK, Jörg: *Sicherheit und Kryptographie im Internet*. 2. Auflage. Wiesbaden : Vieweg & Sohn Verlag, 2005
- [Stein 2007] STEIN, William: *Elementary Number Theory, A Computational Approach*. März 2007. – URL <http://modular.math.washington.edu/ent/ent.pdf>
- [Wikipedia 2008a] WIKIPEDIA: *Bézouts identity* — *Wikipedia, The Free Encyclopedia*. 2008. – URL http://en.wikipedia.org/wiki/Bezouts_identity. – letzter Zugriff am 2. September 2008
- [Wikipedia 2008b] WIKIPEDIA: *Chinesischer Restsatz* — *Wikipedia, Die freie Enzyklopädie*. 2008. – URL http://de.wikipedia.org/wiki/Chinesischer_Restsatz. – letzter Zugriff am 2. September 2008
- [Wikipedia 2008c] WIKIPEDIA: *Division algorithm* — *Wikipedia, The Free Encyclopedia*. 2008. – URL http://en.wikipedia.org/wiki/Division_algorithm. – letzter Zugriff am 2. September 2008
- [Wikipedia 2008d] WIKIPEDIA: *Shor's algorithm* — *Wikipedia, The Free Encyclopedia*. 2008. – URL http://en.wikipedia.org/wiki/Shor%27s_algorithm. – letzter Zugriff am 2. September 2008

Ich erkläre hiermit, dass ich die Facharbeit ohne fremde Hilfe angefertigt und nur die im Literaturverzeichnis angeführten Quellen und Hilfsmittel benutzt habe.

....., den
 (Ort) (Datum) (Unterschrift des Schülers)