Kryptographie

Studienbrief 5

universität wien

Public-Key Verfahren

Gerald und Susanne Teschl

Druckfehler/Feedback bitte an:
gerald.teschl@univie.ac.at

SS 23

Version:
2023-06-30

Studienbrief 5

Public-Key Verfahren

Inhalt

5.1 Public-Key-Verfahren 200

5.2 Zyklische Gruppen 208

5.3 Sicherheit des DH-Schlüsseltausch und des DLP 217

5.3.1 Baby Step-Giant Step 220

5.3.2 Pohlig–Hellman Reduktion 221

5.3.3 Pollard Rho 223

5.3.4 Index Calculus 226

5.3.5 Shor Algorithmus 228

5.4 Der RSA-Verschlüsselungsalgorithmus 228

5.5 Sicherheit des RSA-Algorithmus 233

5.5.1 Sieb des Eratosthenes 240

5.5.2 Fermat 241

5.5.3 $p \pm 1$ Faktorisierungsmethode 242

5.5.4 Pollard Rho 244

5.5.5 Faktorisierung mit dem quadratischen Sieb 244

5.5.6 Shor Algorithmus 246

5.6 Primzahltests 248

5.7 Kontrollfragen 255

5.8 Übungen 262

5.1 Public-Key-Verfahren

Alle bisher besprochenen Verschlüsselungsverfahren haben ein gemeinsames Problem: Immer, wenn Alice und Bob eine geheime Nachricht austauschen möchten, müssen sie vorher einen gemeinsamen geheimen Schlüssel k vereinbaren. Für diesen Schlüsselaustausch muss ein sicherer Weg gefunden werden.

Die Schlüssel der amerikanischen Regierung wurden in den 70er Jahren von einer eigenen Firma verwaltet und über Krypto-Treuhänder verteilt. Auch Banken ließen Schlüssel von besonders zuverlässigen Mitarbeitern verteilen: diese fuhren durchs Land und verteilten Schlüssel an Kunden, die in der Woche darauf Mitteilungen von ihrer Bank erhalten sollten. Mit wachsenden Unternehmensnetzen wuchs sich das aber zu einem logistischen Alptraum aus und war natürlich extrem teuer.

Deshalb begann man darüber nachzudenken, ob es nicht möglich sei, über einen unsicheren Kanal (z.B. das Internet) auf sicherem Weg einen Schlüssel auszutauschen.

1976 schlugen Whitfield Diffie und Martin Hellman [DH76], aufbauend auf Ideen von Ralph Merkle [Mer78], das Konzept von **asymmetrischer Kryptographie** (**Public Key Kryptographie**) vor:

Bei den bisherigen konventionellen Verfahren gibt es immer einen geheimen Schlüssel k, der die Ver- und Entschlüsselungsfunktion E_k bzw. D_k festlegt. Insbesondere kann bei bekannter Verschlüsselungsfunktion E_k auch leicht die zugehörige Entschlüsselungsfunktion D_k berechnet werden. Ein solches Verfahren wird daher auch **symmetrisches Verfahren** genannt.

Bei einem **asymmetrischen Verfahren** besitzt jeder Kommunikationsteilnehmer *zwei* Schlüssel: seinen **öffentlichen Schlüssel** e (**public key**) und seinen privaten Schlüssel d (**private key**). Nur der private Schlüssel d muss geheim gehalten werden, der Schlüssel e kann hingegen öffentlich bekannt gemacht werden (wie eine Telefonnummer in einem Telefonbuch). Wenn Bob an Alice eine geheime Nachricht senden möchte, schlägt er einfach im entsprechenden öffentlichen Verzeichnis Alices öffentlichen Schlüssel e nach, verschlüsselt damit die Nachricht, und sendet sie dann z.B. als Email an Alice. Da nur Alice den zugehörigen privaten Schlüssel d kennt, ist nur sie in der Lage, dieses Email wieder zu entschlüsseln. Wie sieht es mit der Sicherheit aus? Es ist klar, dass der Zusammenhang zwischen der unverschlüsselten und der verschlüsselten Nachricht eindeutig sein muss. Zur bijektiven Verschlüsselungsfunktion E_e existiert also die Umkehrfunktion $D_d = E_e^{-1}$ und ein Angreifer kann diese somit natürlich *theoretisch* ermitteln. Wenn aber für Mallory die Berechnung von d aus e bzw. von $D_d = E_e^{-1}$ aus E_e so langwierig ist, dass er sie auch mit den schnellsten Computern nicht innerhalb praktischer Zeitgrenzen durchführen kann, so ist das Verfahren *praktisch* sicher. Fassen wir zusammen:

Definition 5.1 Ein **Public-Key-Kryptosystem** (PKC) ist eine Menge von Schlüsseln (e, d) mit folgenden Eigenschaften:

- Für jedes Paar (e, d) gibt es zueinander inverse Funktionen E_e und $D_d = E_e^{-1}$, d.h.,
$$(E_e \circ D_d)(x) = (D_d \circ E_e)(x) = x.$$
Mit E_e wird verschlüsselt, mit D_d wird entschlüsselt (engl. *encrypt* = *verschlüsseln* bzw. *decrypt* = *entschlüsseln*).
- Ohne Kenntnis von d kann D_d aus E_e nicht innerhalb praktischer Zeitgrenzen ermittelt werden.

Eine solche Funktion E_e wird **Einweg-Falltürfunktion** (oder **one-way trapdoor function**) genannt: Sie ist leicht auszuführen, aber praktisch nicht umkehrbar (daher *one-way*); es gibt jedoch eine Geheiminformation (daher *trapdoor*) d, mit deren Hilfe man die Funktion leicht umkehren kann. Mit anderen Worten: Verschlüsseln kann jeder leicht, entschlüsseln kann aber nur, wer den geheimen Schlüssel d kennt.

Eine Einweg-Falltürfunktion kann man sich wie einen privaten Postkasten vorstellen. Jeder kann einen Brief leicht einwerfen, heraus bekommt man den Brief aber nicht so leicht („Einwegfunktion“), außer man ist in Besitz des Postkastenschlüssels (entspricht der Geheiminformation d).

Ein solches PKC kann auf einen Schlag auch noch ein weiteres Problem lösen: man kann damit nämlich die Authentizität des Senders und die Unversehrtheit einer Nachricht überprüfen. Bei einem gewöhnlichen Dokument geschieht das ja durch die Unterschrift am Ende. Wie kann eine elektronische Unterschrift aussehen? Mit einem PKC geht das ganz einfach: Wenn Alice ihre Nachricht x digital signieren möchte, wendet sie darauf ihren privaten Schlüssel d an, berechnet also die Signatur $s = D_d(x)$, und versendet dann Nachricht und Signatur gemeinsam, (x, s), an Bob. (Die Nachricht wird im Klartext mitgesendet, es geht ja nun nicht um Geheimhaltung). Bob überprüft mit Alices öffentlichen Schlüssel e, ob er aus der Signatur die Nachricht zurückgewinnt, also ob $E_e(s) = x$. Falls ja, so gilt $s = E_e^{-1}(x) = D_d(x)$, und da nur Alice D_d kennt, kann die Nachricht nur von ihr sein. Zusätzlich kann Bob sicher sein, dass die Nachricht bei der Übertragung nicht verändert wurde. Wir werden später noch auf diese **digitale Signatur** zurückkommen.

Zunächst, als Diffie und Hellman diese Ideen publizierten [DH76], war die Idee eines Public-Key-Verfahrens aber noch ein Luftschloss, denn solch eine Einweg-Falltürfunktion war nicht bekannt.

Eine der ersten Ideen zur Lösung des Problems, wie Alice und Bob über einen unsicheren Kanal ein gemeinsames Geheimnis austauschen können, lautete wie folgt: Alice legt ihre Nachricht in eine Truhe und sichert sie mit einem Vorhängeschloss, zu dem nur sie den Schlüssel a besitzt. Die verschlossene Truhe schickt sie an Bob. Dieser sichert die Truhe *zusätzlich* mit einem zweiten Vorhängeschloss, zu dem nur er den Schlüssel b besitzt und schickt sie zurück an Alice. Alice entfernt ihr Schloss und schickt die Truhe wieder an Bob. Dieser entfernt nun sein

Schloss, und entnimmt die Nachricht. Während die Truhe unterwegs ist (unsicherer Übertragungskanal), ist sie immer durch mindestens ein Schloss gesichert!

No-Key-Protokoll:

Alice		Bob
x, a	$\longrightarrow$ $E_a(x)$	
	$\longleftarrow$ $(E_b \circ E_a)(x)$	b
$(E_a^{-1} \circ E_b \circ E_a)(x) = E_b(x)$	$\longrightarrow$ $E_b(x)$	$(E_b^{-1} \circ E_b)(x) = x$

Leider ließ sich diese Idee nicht sofort praktisch umsetzen: Die Vorhängeschlösser E_a bzw. E_b entsprechen einer Verschlüsselung, d.h., E ist ein Verschlüsselungsverfahren, a und b sind verschiedene Schlüssel. Die entscheidende Eigenschaft der Vorhängeschlösser hier ist, dass sie in beliebiger Reihenfolge angebracht und entfernt werden können. Das verwendete Verschlüsselungsverfahren E müsste $(E_a^{-1} \circ E_b \circ E_a)(x) = E_b(x)$ erfüllen, d.h., die Verschlüsselungsvorschriften von Alice und Bob müssten **kommutieren**

$$(E_a \circ E_b)(x) = (E_b \circ E_a)(x).$$

Das schränkt die möglichen Verschlüsselungsverfahren E natürlich ein. AES und DES scheiden zum Beispiel aus, da sie nicht kommutativ sind. Das One-Time-Pad erfüllt aber zum Beispiel diese Forderung.

Es gibt aber noch ein zweites Problem: Eve, die die Kommunikation belauscht, sieht $E_a(x)$, $E_a(E_b(x)) = E_b(E_a(x))$ und $E_b(x)$. Ist das Verschlüsselungsverfahren E nicht gegen einen *Klartextangriff* sicher, so kann Eve aus $E_a(x)$ („Klartext“) und $E_b(E_a(x))$ („Geheimtext“) den geheimen Schlüssel b von Bob berechnen und dann mit E_b^{-1} die Nachricht x. Somit ist auch das One-Time-Pad für das No-Key-Protokoll nicht verwendbar. Da auch kein anderes Verfahren mit den geforderten Eigenschaften bekannt war, konnte die Idee nicht praktisch umgesetzt werden.

Außerdem hat das Verfahren noch den Nachteil, dass Nachrichten nicht spontan ausgetauscht werden können: Wenn Bob gerade schläft, so kann Alice zwar $E_a(x)$ schicken, muss dann aber warten, bis $E_b(E_a(x))$ kommt, um weitermachen zu können.

Man kann diese Idee jedoch zur Vereinbarung eines geheimen **Sitzungsschlüssels** (= Schlüssel, der nur *einmal* verwendet wird) verwenden, der dann für einen konventionellen symmetrischen Algorithmus verwendet wird. Diffie und Hellman haben dafür in [DH76] folgendes Protokoll vorgeschlagen:

Man legt eine fixe Nachricht g fest. Diese kann für alle Benutzer gleich sein und ist öffentlich bekannt. Wollen Alice und Bob kommunizieren, so schicken sie sich gegenseitig $E_a(g)$ bzw. $E_b(g)$. Nun berechnet Alice $y_a = E_a(E_b(g))$ und Bob $y_b = E_b(E_a(g))$. Aus der Kommutativität folgt $y_a = y_b$ und Alice und Bob haben somit ein gemeinsames Geheimnis $k = y_a = y_b$, das sie als Schlüssel für einen konventionellen Algorithmus verwenden können:

Diffie–Hellman-Schlüsselaustausch:

Alice	g ist öffentlich bekannt	Bob
a	$\longrightarrow$ $E_a(g)$	b
	$\longleftarrow$ $E_b(g)$	
$k = (E_a \circ E_b)(g)$		$k = (E_b \circ E_a)(g)$

Möchte Eve, die $E_a(g)$ bzw. $E_b(g)$ belauscht, an dieses Geheimnis k kommen, so ist eine Möglichkeit mit einem Klartextangriff den geheimen Schlüssel von Alice (da sie ja g und $E_a(g)$ kennt) oder Bob zu ermitteln. Dagegen muss das verwendete Verfahren E also sicher sein. Gibt es außer diesem Weg keinen anderen (leichteren) Weg, um an die geheimen Schlüssel von Alice und Bob zu kommen, so ist das Verfahren sicher.

Für diese Schlüsselvereinbarung nach Diffie–Hellman ist es nicht notwendig, dass die Funktion E invertierbar ist (was für ein Verschlüsselungsverfahren notwendig ist, da sonst nicht entschlüsselt werden kann).

Nun ist insbesondere auch spontaner Nachrichtenaustausch möglich: Alice kann $E_a(g)$ als ihren öffentlichen Schlüssel jedermann bekannt geben. Möchte Bob eine Nachricht an Alice schicken, so berechnet er aus dem öffentlichen Schlüssel $E_a(g)$ den Sitzungsschlüssel $k = E_b(E_a(g))$, verschlüsselt damit die Nachricht nach einem konventionellen symmetrischen Verfahren, und schickt das Ergebnis zusammen mit seinem öffentlichen Schlüssel $E_b(g)$ an Alice. Alice berechnet aus $E_b(g)$ den Sitzungsschlüssel und entschlüsselt die Nachricht.

Wenn man das Diffie–Hellman–Protokoll zum Schlüsselaustausch verwendet, dann sollten a und b für jede Sitzung neu erzeugt werden (und danach vernichtet). Sonst könnte ein Angreifer, der die verschlüsselte Kommunikation mitschneidet und im Nachhinein an a oder b kommt, die gespeicherte Kommunikation entschlüsseln. Dies wird durch das Neu-Erzeugen von a bzw. b verhindert. Man spricht von **Perfect Forward Secrecy (PFS)**. Im Internet trat dieses Problem beim **Heartbleed Bug** auf, durch den geheime Schlüssel von anfälligen Webservern ausgelesen werden konnten. Ohne PFS konnte jeder (Geheimdienst), der die Kommunikation mitgeschnitten hat, auch die vergangene Kommunikation entschlüsseln. Da das PKC auch zur Überprüfung der Authentizität der Komminikationsteilnehmer mittels digitaler Signatur (siehe Abschnit 6.1) zum

Einsatz kommt, hat man früher aus Performance-Gründen auf den Einsatz eines weiteren Verfahrens zum Schlüsseltausch verzichtet. Deshalb war PFS auf vielen Webservern nicht aktiviert.

Damit war die Public-Key-Kryptographie geboren und alles, was noch fehlte, war eine geeignete Funktion E_a. Sie muss nicht invertierbar sein, es genügt, wenn sie eine **Einwegfunktion** (*one-way function*) in Bezug auf a ist. Sie muss aber die Eigenschaft haben, dass $E_a \circ E_b = E_b \circ E_a$ gilt. Auch dafür fanden Diffie und Hellman eine Lösung: die Exponentialfunktion in $\mathbb{Z}_p^*$:

$$y = g^a \pmod{p}.$$

Da es eine Funktion von $\mathbb{Z}_p^*$ nach $\mathbb{Z}_p^*$ ist, sie also nur endliche viele Funktionswerte annimmt, wird sie auch **diskrete Exponentialfunktion** genannt. Sie ist, im Gegensatz zu ihrer Umkehrfunktion, der diskreten Logarithmusfunktion, leicht berechenbar, also eine Einwegfunktion.

Beispiel 5.2 Diskrete Exponentialfunktion
Stellen Sie eine Wertetabelle für $y = 7^a \,(\mathrm{mod}\, 11)$ auf.

Lösung zu 5.2 a durchläuft alle Zahlen aus $\mathbb{Z}_{11}^*$; wir berechnen den Rest bereits in Zwischenschritten, damit die Zahlen nicht zu groß werden, z.B. $7^4 = 7^2 \cdot 7^2 = 49 \cdot 49 = 5 \cdot 5 = 25 = 3 \,(\mathrm{mod}\, 11)$

a	1	2	3	4	5	6	7	8	9	10
$y = 7^a \,(\mathrm{mod}\, 11)$	7	5	2	3	10	4	6	9	8	1

■

Also sieht der **Diffie–Hellman Schlüsselaustausch** (Diffie–Hellman Schlüsselvereinbarung, *Diffie–Hellman Key Exchange*, DHKE) wie folgt aus:

- Alice und Bob einigen sich auf eine Primzahl p und eine Zahl $g \in \mathbb{Z}_p^*$. (Diese Zahlen können für alle Benutzer gleich gewählt werden.)
- Alice wählt eine geheime Zahl a und Bob eine geheime Zahl b ($1 < a, b < p - 1$).
- Alice schickt $\alpha = g^a \pmod{p}$ an Bob. Bob schickt $\beta = g^b \pmod{p}$ an Alice.
- Alice berechnet $\beta^a \pmod{p}$, Bob berechnet $\alpha^b \pmod{p}$. Wegen

  $$\beta^a = g^{b \cdot a} = g^{a \cdot b} = \alpha^b \pmod{p}$$

  sind beide Zahlen gleich und können als geheimer Schlüssel (z.B. für AES) verwendet werden.

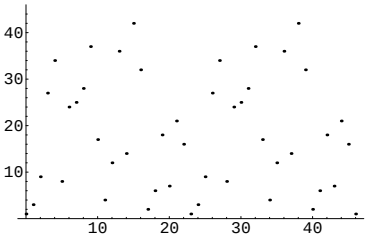


Abbildung 5.1: Diskrete Exponentialfunktion $3^a \pmod{47}$

Beispiel 5.3 Diffie–Hellman Schlüsselaustausch
Alice und Bob einigen sich auf $g = 7$, $p = 11$; Alice wählt als geheime Zahl $a = 2$ und Bob wählt $b = 4$. Wie lautet das gemeinsame Geheimnis?

Lösung zu 5.3 (Alle Berechnungen sind hier modulo 11 zu verstehen, das wird nicht mehr explizit dazugeschrieben.) Alice schickt an Bob $\alpha = g^a = 7^2 = 5$, Bob berechnet daraus $k = \alpha^b = 5^4 = 9$. Alice bekommt dasselbe Ergebnis: von Bob erhält sie $\beta = g^b = 7^4 = 3$, und daraus berechnet sie $k = \beta^a = 3^2 = 9$. ■

Worauf beruht die Sicherheit der Diffie–Hellman Schlüsselvereinbarung? Eve kennt g und p, und kann die Werte α und β belauschen. Der einzige bekannte Weg für Eve, um an das gemeinsame Geheimis zu kommen, ist den geheimen Schlüssel von Alice (oder Bob) zu ermitteln. Dazu muss Eve die Gleichung

$$\alpha = g^a \pmod{p}$$

nach a auflösen. Sie muss also den **diskreten Logarithmus** von α zur Basis g in $\mathbb{Z}_p^*$ berechnen. Im Prinzip ist das natürlich immer möglich, Eve braucht ja nur Werte $a \in \mathbb{Z}_p^*$ durchzuprobieren, bis die Gleichung erfüllt ist (Brute Force Angriff). Der Haken dabei ist aber, dass die *diskrete* Exponential- und Logarithmusfunktion (im Gegensatz zur *reellen* Exponential- und Logarithmusfunktion) nicht stetig ist, sondern unvorhersagbare Sprünge macht, was wir uns auch graphisch leicht veranschaulichen können: Abbildung 5.1. Es ist zwar einfach, zu gegebenem a den Wert $\alpha = 3^a$ modulo 47 zu berechnen, aber extrem schwer, von gegebenem α auf den Wert $a = \log_3 \alpha$ zu schließen (Einwegeigenschaft der diskreten Exponentialfunktion). Es ist für die Angreiferin Eve *praktisch nicht möglich*, a zu finden, sofern die Primzahl p groß genug ist.

Um die Sicherheit des DHKE zu gewährleisten, muss man sicherstellen, dass das Problem des diskreten Logarithmus nicht effektiv gelöst werden kann:

Definition 5.4 (Diskretes Logarithmusproblem (DLP)) Sei $g, \alpha \in \mathbb{Z}_p^*$. Das DLP ist das Problem, jene natürliche Zahl x zu finden, für die

$$g^x = \alpha \pmod{p}$$

ist.

Wir werden uns damit in Abschnitt 5.3 näher beschäftigen.

Obwohl der Algorithmus von Diffie–Hellman das Problem des Schlüsseltausch über einen unsicheren Kanal befriedigend löst, ist dieses genaugenommen noch kein PKC nach Definition 5.1, da Alice mit ihrem privaten Schlüssel keine Nachrichten verschlüsseln kann, die jeder mit dem öffentlichen Schlüssel entschlüsselt kann. Die zu Beginn beschriebene digitale Signatur ist damit z.B. nicht möglich. Die Suche nach einer Einweg-*Falltür*funktion ging daher weiter. Zwei Jahre später wurde eine Einweg-Falltürfunktion gefunden – dies ist Thema des nächsten Abschnitts.

Allerdings kann Alice ihren Teil beim DH-Schlüsseltausch vorberechnen und als öffentlichen Schlüssel bekannt geben. Einigt man sich wie das geteilte Geheimnis als Schlüssel für einen symmetrischen Algorithmus zu verwenden ist, so kann jeder an Alice eine geheime Nachricht schicken, die nur sie entschlüsseln kann. Dieses Verfahren wurde 1984 vom ägyptischen Kryptologen Taher Elgamal vorgeschlagen und ist als **Elgamal-Verschlüsselungsalgorithmus** bekannt:

- Alice und Bob einigen sich auf eine Primzahl p und eine Zahl $g \in \mathbb{Z}_p^*$. (Diese Zahlen können für alle Benutzer gleich gewählt werden und sind öffentlich bekannt.)

- Alice wählt eine geheime Zahl a $(1 < a < p-1)$ und gibt ihren öffentlichen Schlüssel $\alpha = g^a \pmod p$ bekannt.

- Bob wählt eine zufällige Zahl b $(1 < b < p-1)$, berechnet $k = \alpha^b \pmod p$ und verschlüsselt damit die Nachricht x gemäß $y = kx \pmod p$. Bob berechnet $\beta = g^b \bmod p$ und schickt die verschlüsselte Nachricht (y, β) an Alice.

- Alice berechnet $k = \beta^a = g^{a \cdot b} \pmod p$ und das zugehörige multiplikative Inverse k^{-1} und entschlüsselt damit die Nachricht $x = k^{-1}y \pmod p$. Nach dem kleinen Satz von Fermat ist das multiplikative Inverse hier durch $k^{-1} = \beta^{p-1-a} \pmod p$ gegeben.

Elgamal-Verschlüsselung

Alice	g und p sind öff. bekannt	**Bob**
a, $\alpha = g^a \pmod p$	α ist öff. Schl. von Alice	
		Nachricht x Zufallszahl b, $\beta = g^b \pmod p$ $k = \alpha^b \pmod p$ $y = k \cdot x \pmod p$
	$\longleftarrow$ c, β	
$k = \beta^a \pmod p$ $k^{-1} = \beta^{p-1-a} \pmod p$ $x = k^{-1}y \pmod p$		

Die Verschlüsselungsvorschrift $y = k \cdot x \pmod p$ ist eine bijektive Abbildung von $\mathbb{Z}_p^*$ nach $\mathbb{Z}_p^*$ und die Entschlüsselungsvorschrift ist die zugehörige Umkehrabbildung. Wenn die zu verschlüsselnde Nachricht x eine Zahl größer p ist, so muss sie zuvor in mehrere Zahlen (Blöcke) $x_1, \ldots, x_N$ kleiner p umgewandelt werden. Da das verwendete Verfahren nur eine einfache Multiplikationschiffre ist, ist es wesentlich, dass Bob für jede Nachricht x_i eine andere geheime Zahl b_i (**Nonce**, *number used once*) wählt. Denn wenn alle Nachricht mit demselben b und somit mit demselben Schlüssel k verschlüsselt würden, dann könnte ein Angreifer einen Klartextangriff machen: aus einem bekannten zusammengehörenden Stück Klartext x/Geheimtext y, kann der zugehörige Schlüssel über $k = yx^{-1} \pmod p$ berechnet werden!

Beispiel 5.5 Elgamal-Verschlüsselung
$g = 3$, $p = 7$, der öffentliche Schlüssel von Alice sei $\alpha = 4$.
a) Verschlüsseln Sie die Nachricht 123.
b) Entschlüsseln Sie $(y, \beta) = (4, 3)$ mithilfe von Alices geheimen Schlüssel $a = 4$.

Lösung zu 5.5

a) Um $x = 1$ zu verschlüsseln, wählt Bob eine Zufallszahl, z.B. $b = 4$; er berechnet damit den Schlüssel $k = \alpha^b = 4^4 = 4 \pmod 7$, und verschlüsselt damit die Nachricht zu $c = k \cdot x = 4 \cdot 1 = 4 \pmod 7$. Dann berechnet er $\beta = g^b = 4 \pmod 7$, und schickt diese Zahl gemeinsam mit der verschlüsselten Nachricht c an Alice: $(y, \beta) = (4, 4)$. Dann wählt Bob für $x = 2$ und $x = 3$ zwei weitere Zufallszahlen, z.B. $b = 5$ bzw. $b = 2$, und berechnet damit analog $y = 4$ bzw. $y = 6$ und $\beta = 5$ bzw. $\beta = 2$ und sendet an Alice $(4, 5)$, $(6, 2)$. Insgesamt lautet die verschlüsselte Nachricht also $(4, 4)$, $(4, 5)$, $(6, 2)$.

b) Alice berechnet den Schlüssel $k = \beta^a = 3^4 = 4 \pmod 7$. Dann berechnet sie $k^{-1} = \beta^{p-1-a} = 3^{7-1-4} = 2 \pmod 7$. Damit entschlüsselt sie $x = k^{-1}y = 2 \cdot 4 = 1 \pmod 7$.

■

Ein zugehöriges Signaturverfahren, der **Elgamal-Signaturalgorithmus**, wurde ebenfalls vorgeschlagen. Wir werden darauf in Abschnitt 6.1 zurückkommen.

5.2 Zyklische Gruppen

Grob gesagt wirft das DLP folgende mathematische Fragen auf:

- Kann man die Potenzen g^j überhaupt effektiv berechnen?
- Wie viele verschiedene Elemente hat die Folge der Potenzen g, g^2, g^3, ... (das ist die Anzahl der Elemente die bei einem Brute-Froce-Angriff durchsucht werden müssen)?
- Wie berechnet man diese Anzahl bzw. wie stellt man sicher, dass sie möglichst groß wird? Oder anders gefragt, was ist die maximale Anzahl und wie viele Elemente realisieren diese?

Diese Fragen können nicht nur in $\mathbb{Z}_p$ sondern über einer beliebigen Gruppe betrachtet werden und diese Verallgemeinerung spielt eine wichtige Rolle in die Kryptographie. Wollen wir diese Zusammenhänge besser verstehen ist ein kleiner Ausflug in die abstrakte Algebra notwendig.

Gruppen bzw. Untergruppen haben wir bereits kennengelernt (siehe Definition 1.37). Für jedes $m \in \mathbb{N}$ ist z.B. $(\mathbb{Z}_m^*, \cdot)$ eine Gruppe, ebenso $(\mathbb{Z}_m, +)$. Wenn wir im Folgenden kurz von der Gruppe $\mathbb{Z}_m^*$ sprechen, so ist immer die Gruppe $(\mathbb{Z}_m^*, \cdot)$ gemeint (also die Menge $\mathbb{Z}_m^*$ mit der Multiplikation modulo m); analog meinen wir, wenn wir kurz von der Gruppe $\mathbb{Z}_m$ sprechen, die Gruppe $(\mathbb{Z}_m, +)$.

Beispiel 5.6 Untergruppen
Geben Sie alle Untergruppen von $\mathbb{Z}_7^*$ an.

Lösung zu 5.6 Nach Satz 1.38 gilt: Eine Teilmenge von $\mathbb{Z}_7^*$ ist eine Untergruppe, wenn sie abgeschlossen ist bzüglich der Multiplikation $\cdot$ (modulo 7) und zu jedem Element auch dessen Kehrwert enthält. Insbesondere muss in jeder Untergruppe das neutrale Element (hier 1) enthalten sein. Mithilfe der Multiplikationstabelle („scharfer Blick“ :-) finden wir demnach vier Untergruppen: $U = \{1\}$, $V = \{1, 6\}$, $W = \{1, 2, 4\}$, $\mathbb{Z}_7^*$. ■

Wir bemerken hier etwas: Die Untergruppen haben entweder 1 oder 2 oder 3 oder (die ganze Gruppe als Untergruppe betrachtet) 6 Elemente. Diese Zahlen sind genau die Teiler der Gruppenordnung 6. Das ist kein Zufall:

Satz 5.7 (Satz von Lagrange) G sei eine endliche Gruppe, H sei Untergruppe. Dann teilt die Ordnung von H (also die Anzahl der Elemente von H) die Ordnung von G. D.h. $|H|$ ist Teiler von $|G|$.

Um das zu sehen betrachtet man die Äquivalenzrelation $a \equiv b$ genau dann wenn $ab^{-1} \in H$ (Reflexivität, Symmetrie und Transitivität sind leicht zu sehen). Also haben wir eine Partition von G in die zugehörigen Äquivalenzklassen, die Nebenklassen, $aH = \{ac | c \in H\}$. Da jede Nebenklasse $|aH| = |H|$ Elemente hat (und da Aquivalenzklassen disjunkt sind) folgt $|G| = k|H|$, wobei k die Anzahl der verschiedenen Nebenklassen bezeichnet.

Grundsätzlich muss es nicht zu jedem Teiler auch eine Untergruppe geben. Weiters könnte es zu einem Teiler mehr als eine Untergruppe geben.

Eine einfache Möglichkeitkeit, eine Untergruppe zu erzeugen, ist es, ein gegebenes Element $a \in G$ immer wieder mit sich selbst zu verknüpfen. Dazu führen wir rekursiv folgende Abkürzung ein

$$a^0 = n, \qquad a^{j+1} = a \circ a^j, \quad j \in \mathbb{N},$$

also

$$a^j = \underbrace{a \circ a \circ a \circ \ldots \circ a}_{j \text{ mal}}.$$

Für negatives j erweitert man diese Definition zu

$$a^{-j} = i(a^j) = i(a)^j$$

(mit $i(.)$ dem inversen Element), sodass die üblichen **Potenzgesetzte**

$$a^j \circ a^k = a^{j+k}, \qquad j, k \in \mathbb{Z}$$

gelten.

In einer kommutativen Gruppe schreibt man oft + für die Gruppenverknüpfung und definiert demenstprechend $0a = 0$ (die linke Null ist hier $0 \in \mathbb{Z}$ und die rechte Null ist $0 = n \in G$, das neutrale Element in G) bzw. $(j+1)a = j\,a + a$ und $(-j)a = -(ja)$, $j \in \mathbb{N}$.

Betrachten wir z.B. $2 \in \mathbb{Z}_5^*$, so gilt

$$2, \quad 2^2 = 4, \quad 2^3 = 3, \quad 2^4 = 1, \quad 2^5 = 2, \ldots$$

und es ist klar, dass sobald wir $2^4 = 1$ (das neutrale Element) erreicht haben, sich die Folge periodisch wiederholt. Das muss bei einer endlichen Gruppe immer passieren. Betrachten wir dazu die Folge a^j, $j \in \mathbb{N}_0$. Dann muss es aufgrund der Endlichkeit von G in dieser Folge irgendwann zu einer Kollision kommen, es muss also $a^j = a^k$ für $j < k$ gelten. Formen wir diese Gleichung um, so sehen wir, dass

sogar $a^{k-j} = n$ gelten muss. Es gibt also ein kleinstes $k \in \mathbb{N}$ mit der Eigenschaft $a^k = n$ und ab k wiederholt sich die Folge periodisch. Die Werte $n, a, a^2, \cdots, a^{k-1}$ dieser Folge sind also abgeschlossen bezüglich der Gruppenverknüpfung und wegen $(a^j)^{-1} = n \circ a^{-j} = a^k \circ a^{-j} = a^{k-j}$ auch abgeschlossen bezüglich Bildung des Inversen.

Definition 5.8 $(G, \circ)$ sei eine endliche Gruppe mit neutralem Element n. Die **Ordnung eines Elementes** $a \in G$ ist die kleinste natürliche Zahl $k \in \mathbb{N}$ mit

$$a^k = n.$$

Schreibweise: $\mathrm{ord}(a) = k$. Die von a erzeugte Untergruppe bezeichnet man mit

$$\langle a \rangle = \{n, a, a^2, \cdots, a^{k-1}\}$$

und es gilt $|\langle a \rangle| = \mathrm{ord}(a)$.

Die Behauptung $|\langle a \rangle| = \mathrm{ord}(a)$ folgt, da es in der Auflistung keine Kollisionen mehr geben kann, denn sonst hätten wir eine kleinere Ordnung: Aus $a^j = a^k$ folgt ja $a^{j-k} = n$ und ist $0 \leq j - k < \mathrm{ord}(a)$, so muss $j = k$ sein.

Offensichtlich ist $\langle a \rangle$ wegen $a^j \circ a^k = a^{j+k} = a^{k+j} = a^k \circ a^j$ kommutativ. Außerdem können wir beim Potenzieren immer den Exponenten modulo der Ordnung reduzieren,

$$a^j = a^{j \bmod \mathrm{ord}(a)},$$

und es gilt

$$a^j = a^k \quad \Leftrightarrow \quad j = k \pmod{\mathrm{ord}(a)}.$$

Beispiel 5.9 Ordnung
Geben Sie für jedes Element aus
a) $G = \mathbb{Z}_7^*$ b) $G = \mathbb{Z}_8^*$ c) $G = \mathbb{Z}_{10}^*$
die Ordnung und die von diesem Element erzeugte Untergruppe an.

Lösung zu 5.9 a) Wir schreiben für jedes $a \in \mathbb{Z}_7^* = \{1, 2, \ldots, 6\}$ alle möglichen Potenzen an, woraus wir alles weitere ablesen können:

a	a^1	a^2	a^3	a^4	a^5	a^6	$\mathrm{ord}(a)$	$\langle a \rangle$
1	1	1	1	1	1	1	1	$\{1\}$
2	2	4	1	2	4	1	3	$\{1, 2, 4\}$
3	3	2	6	4	5	1	6	$\mathbb{Z}_7^*$
4	4	2	1	4	2	1	3	$\{1, 2, 4\}$
5	5	4	6	2	3	1	6	$\mathbb{Z}_7^*$
6	6	1	6	1	6	1	2	$\{1, 6\}$

Wir sehen, dass die Elemente $a = 3$ und $a = 5$ mit ihren Potenzen die ganze Gruppe *erzeugen*.
b) $\mathbb{Z}_8^* = \{1, 3, 5, 7\}$. Da $|\mathbb{Z}_8^*| = 4$,

a^1	a^2	a^3	a^4	ord(a)	$\langle a \rangle$
1	1	1	1	1	$\{1\}$
3	1	3	1	2	$\{1, 3\}$
5	1	5	1	2	$\{1, 5\}$
7	1	7	1	2	$\{1, 7\}$

Hier hat kein Element die (maximal mögliche) Ordnung 4. Somit erzeugt kein Element $\mathbb{Z}_8^*$.
c) $\mathbb{Z}_{10}^* = \{1, 3, 7, 9\}$, somit $|\mathbb{Z}_{10}^*| = 4$.

a^1	a^2	a^3	a^4	ord(a)	$\langle a \rangle$
1	1	1	1	1	$\{1\}$
3	9	7	1	4	$\mathbb{Z}_{10}^*$
7	9	3	1	4	$\mathbb{Z}_{10}^*$
9	1	9	1	2	$\{1, 9\}$

Wiederum erzeugen 3 und 7 die ganze Gruppe. ■

Wir sehen übrigens an den Tabellen, dass in der letzten Spalte immer das neutrale Element steht. Dies ist kein Zufall:

Satz 5.10 In jeder endlichen Gruppe $(G, \circ)$ ist die Ordnung eines Elementes $a \in G$ immer ein Teiler der Gruppenordnung und es gilt (n bezeichnet das neutrale Element von G):

$$a^{|G|} = n$$

für beliebiges $a \in G$.

Denn wegen des Satzes von Lagrange ist $|G| = \text{ord}(a) \cdot t$ mit einem $t \in \mathbb{N}$ und somit: $a^{|G|} = a^{\text{ord}(a) \cdot t} = (a^{\text{ord}(a)})^t = n^t = n$.

Im Spezialfall $G = \mathbb{Z}_m$ erhalten wir den Satz von Euler (Satz 1.33) und ist m eine Primzahl, so erhalten wir den Kleinen Satz von Fermat (Satz 1.35).

Wie wir im letzten Beispiel gesehen haben, kann es sein, dass uns $\langle a \rangle$ die ganze Gruppe liefert.

Definition 5.11 Eine endliche Gruppe $(G, \circ)$ heißt **zyklisch**, wenn sie (mindestens) ein Element g mit maximaler Ordnung, das heißt ord$(g) = |G|$, besitzt. Ein Element mit maximaler Ordnung nennt man **Generator** (oder **Primitivwurzel,**

primitives Element, erzeugendes Element, engl. *generator, primitive root*). Für einen Generator g gilt also

$$G = \langle g \rangle,$$

in diesem Sinn wird die Gruppe durch g erzeugt.

Ist $|G|$ eine Primzahl, so gibt es keine nichttrivialen Teiler, es muss also immer $\langle a \rangle = \{n\}$ (falls $a = n$) oder $\langle a \rangle = G$ gelten:

Satz 5.12 Sei G eine endliche Gruppe, deren Ordnung $|G|$ eine Primzahl ist. Dann ist G zyklisch und jedes Element (außer dem neutralen) ist Generator.

Ist G zyklisch mit Generator $g \in G$, so liefert die Abbildung

$$\psi_g(j) = g^j, \qquad 0 \leq j < \mathrm{ord}(g) = |G|,$$

eine bijektive Auflistung der Element von G. Mehr noch, aufgrund von $\psi_g(j+k) = g^{j+k} = g^k \circ g^j = \psi_g(j)\psi_g(k)$ überträgt ψ_g die Gruppenstruktur von $(\mathbb{Z}_{|G|}, +)$ auf $(G, \circ)$. Es gilt also

Satz 5.13 Eine zyklische Gruppe $(G, \circ)$ ist isomorph zu $(\mathbb{Z}_{|G|}, +)$.

Das bedeutet, dass $\mathbb{Z}_m$ die prototypische zyklische Gruppe ist und wir die Eigenschaften von G anhand von $\mathbb{Z}_{|G|}$ verstehen können.

Beispiel 5.14 Berechnen Sie die Ordnung eines beliebigen Elements $a \in (\mathbb{Z}_m, +)$. Wann ist a ein Generator bzw. wie viele Generatoren gibt es?

Lösung zu 5.14 Es gilt $j \cdot a = 0 \pmod{m}$ genau dann wenn j ein Vielfaches von $\frac{m}{\mathrm{ggT}(a,m)}$ ist. Insbesondere ist $\mathrm{ord}(a) = \frac{m}{\mathrm{ggT}(a,m)}$ und somit $\mathrm{ord}(a) = m$ genau dann wenn $\mathrm{ggT}(a, m) = 1$. Es gibt also $\varphi(m)$ Generatoren. ■

Unser Wissen über $\mathbb{Z}_m$ können wir mithilfe von ψ_g sofort auf G übertragen. Da ψ_g bijektiv ist und die Gruppenoperationen erhält, ist $\psi_g(j)$ genau dann ein Generator in G, wenn j ein Generator in $\mathbb{Z}_m$ ist.

Satz 5.15 $(G, \circ)$ sei eine zyklische Gruppe mit Generator g. Dann ist die Menge der Generatoren gegeben durch $\{g^j | j \in \mathbb{Z}^*_{|G|}\}$. Insbesondere gibt es genau $\varphi(|G|)$ Generatoren.

Beispiel 5.16 Wie viele Generatoren besitzen die zyklischen Gruppen
a) $\mathbb{Z}_7^*$ b) $\mathbb{Z}_{10}^*$ c) $(\mathbb{Z}_4, +)$?

Lösung zu 5.16 a) $\mathbb{Z}_7^*$ hat 6 Elemente und besitzt somit $\varphi(6) = 2$ Generatoren (wir haben sie in Beispiel 5.9 explizit gefunden).
b) $\mathbb{Z}_{10}^*$ hat 4 Elemente, hat daher $\varphi(4) = 2$ Generatoren (siehe auch Beispiel 5.9).
c) $(\mathbb{Z}_4, +)$ hat 4 Elemente, also $\varphi(4) = 2$ Generatoren (die beiden Generatoren sind 1 und 3). ■

Die Formel für die Ordnung aus Beispiel 5.14 können wir auch mit ψ übertragen:

Satz 5.17 Ist $(G, \circ)$ eine endliche Gruppe und $a \in G$, so gilt

$$\operatorname{ord}(a^j) = \frac{\operatorname{ord}(a)}{\operatorname{ggT}(\operatorname{ord}(a), j)}.$$

Wir können ohne Beschränkung der Allgemeinheit $G = \langle a \rangle$ voraussetzten, da wir diese Untergruppe nicht verlassen. Betrachte $\psi_a : \mathbb{Z}_m \to G$ mit $m = \operatorname{ord}(a)$. Dann gilt $\psi_a(j) = a^j$ und somit $\operatorname{ord}(a^j) = \operatorname{ord}_{\mathbb{Z}_m}(j) = \frac{m}{\operatorname{ggT}(m,j)}$.

Eine weiter nützliche Tatsache ist

Satz 5.18 Ist $(G, \circ)$ eine endliche kommutative Gruppe und $a, b \in G$ mit $\operatorname{ord}(a) = k$, $\operatorname{ord}(b) = l$ und $\operatorname{ggT}(k, l) = 1$, dann gilt

$$\operatorname{ord}(ab) = kl.$$

Wir haben auf jeden Fall $(a \circ b)^{kl} = (a^k)^l \circ (b^l)^k = n$, also ist $m = \operatorname{ord}(ab)$ ein Teiler von kl. Umgekehrt gilt $n = (a \circ b)^{mk} = b^{km}$ also teilt l das Produkt km. Da l und k keine gemeinsamen Teiler haben, teilt l also m. Analog sehen wir, dass k ebenfalls m teilt.

Wie schon erwähnt, muss es nicht zu jedem Teiler der Gruppenordnung eine Untergruppe geben. Weiters kann es zu einem Teiler auch mehr als eine Untergruppe geben. Bei zyklischen Gruppen können wir das wieder anhand von $\mathbb{Z}_m$ untersuchen.

Beispiel 5.19 Welche Untergruppen gibt es in $(\mathbb{Z}_m, +)$?

Lösung zu 5.19 Ist $H \neq \{0\}$ eine Untergruppe, dann können wir $g = \min\{0 < a < m | a \in H\}$ betrachten. Dann muss auf jeden Fall $\langle g \rangle \subset H$ gelten. Angenommen

es gäbe ein $a \in H \setminus \langle g \rangle$. Dann können wir $h = k \cdot g + j$ mit $0 \leq j < g$ schreiben. Dann ist aber $j = h - k \cdot g \in H$ im Widerspruch zur Minimalität von g. Also gilt $H = \langle g \rangle$. Jede Untergruppe ist also zyklisch und umgekehrt gibt es zu jedem Teiler $t \mid m$ eine Untergruppe die von $g = m/t$ erzeugt wird. ■

Da unser Isomorphismus ψ_g Untergruppen aufeinander abbildet folgt:

Satz 5.20 Sei $(G, \circ)$ eine zyklische Gruppe mit neutralem Element n und Generator g. Dann gibt es *zu jedem Teiler t von $|G|$ genau eine* zyklische Untergruppe H_t mit t Elementen die von $g^{|G|/t}$ erzeugt wird. H_t besteht genau aus jenen $a \in G$ mit $a^t = n$.

Alle Untergruppen sind von dieser Form und insbesondere ist jede Untergruppe zyklisch.

Insbesondere ist die Menge $H_t = \{x \in G | x^t = n\}$ immer eine Untergruppe mit $\text{ggT}(|G|, t)$ Elementen. Denn ist t ein Teiler von $|G|$, so folgt das aus unserem Satz, und anderenfalls ist $H_t = \{n\}$, da t ein Vielfaches von $\text{ord}(x)$ sein muss und somit nur noch $\text{ord}(x) = 1$ in Frage kommt.

Beispiel 5.21 $\mathbb{Z}_{11}^*$ hat 10 Elemente. Die Teiler von 10 sind: 1, 2, 5, 10. Da $\mathbb{Z}_{11}^*$ zyklisch ist, gibt es jeweils genau eine Untergruppe mit 1, 2, 5, bzw. 10 Elementen: $\{1\}$ wird erzeugt von 1. $\{1, 10\}$ wird erzeugt von 10 (einziges Element mit Ordnung 2). $\{1, 3, 4, 5, 9\}$ wird erzeugt von jedem Element mit Ordnung 5, also von jedem der Elemente $3, 4, 5, 9$. $\mathbb{Z}_{11}^*$ wird erzeugt von jedem Element mit Ordnung 10, also von jedem der Elemente $2, 6, 7, 8$.

Wie sieht es nun aus, wenn wir unser Wissen über $\mathbb{Z}_m$ nicht nur theoretisch, sondern auch praktisch verwenden möchten? In die Richtung $\mathbb{Z}_m \to G$ ist ψ_g leicht zu berechnen. Aber in die umgekehrte Richtung $G \to \mathbb{Z}_m$ ist ψ_g^{-1} schwer zu berechnen! Das Problem zu gegebenen $a, b \in G$ ein $j \in \mathbb{Z}$ mit

$$a^j = b$$

zu finden ist das **diskretes Logarithmus-Problem** (**DLP**) für G. Das zugehörige j wird als diskreter Logarithmus $\log_a(b)$ oder auch **Index** $\text{ind}_a(b)$ bezeichnet. Aus den Potenzgesetzten folgt

$$\log_a(n) = 0, \qquad \log_a(b \circ c) = \log_a(b) + \log_a(c), \qquad \log_a(b^j) = j \log_a(b).$$

Beispiel 5.22 Wie kann das DLP $b = j \cdot a$ in $\mathbb{Z}_m$ gelöst werden?

Lösung zu 5.22 Wir suchen eine Lösung der Gleichung $b = ja + km$. Eine Lösung kann es offenbar nur geben, wenn $\text{ggT}(a, m)$ ein Teiler von b ist, wenn also $b = t\text{ggT}(a, m)$. In diesem Fall können wir eine Lösung der Gleichung $\text{ggT}(a, m) = \tilde{j}a + \tilde{k}m$ mithilfe des erweiterten Euklid'schen Algorithmus bestimmen und die gesucht Lösung ist dann $j = t\tilde{j}$. ■

Um das besser zu verstehen, beschäftigen wir uns zunächst mit der direkten Richtung: Wollen wir z.B. a^{17} berechnen, so kann man natürlich mit a starten und dann sechzehn mal mit a multiplizieren. Man kann aber die Berechnung durch wiederholtes Quadrieren effizienter machen:

$$a^{17} = a^{16} \cdot a^1 = a^{2\cdot 2\cdot 2\cdot 2} \cdot a = (((a^2)^2)^2)^2 \cdot a$$

Anstelle von 16 brauchen wir nun nur noch fünf Multiplikationen. Dieses Verfahren heißt Square-and-Multiply-Verfahren, weil man dabei nur quadrieren und multiplizieren muss:

Satz 5.23 (Square-and-Multiply) Sei

$$j = (j_k \cdots j_1 j_0)_2 = j_k 2^k + \cdots + j_1 2 + j_0$$

die Binärdarstellung von j, dann berechnen wir rekursiv

$$a_n = a^{j_n}, \quad a_{n-1} = a_n^2 \circ a^{j_{n-1}}, \quad \cdots, \quad a_0 = a_1^2 a^{j_0},$$

wobei im letzten Schritt $a_0 = a^j$ gilt.

Es wird also mit n gestartet und die Binärdarstellung von j beginnend mit dem höchsten Bit abgearbeitet, wobei in jedem Schritt das bisherige Resultat quadriert wird und, falls das Bit gleich 1 ist, noch zusätzlich mit a multipliziert wird.

Beispiel 5.24 Berechnen Sie 3^6 in $\mathbb{Z}_5$ mit dem Square-and-Multiply-Verfahren.

Lösung zu 5.24 Es gilt $5 = (110)_2$ und wir starten mit $a_2 = a^{j_2} = 3^1 = 3$. Der nächste Wert ist $a_1 = a_2^2 a^{j_1} = 3^2 3^1 = 4 \cdot 3 = 2$. Der letzte Wert ist $a_0 = a_1^2 a^{j_0} = 2^2 1 = 4$. ■

Als Nächstes betrachten wir die Ordnung. Da die Ordnung eines Elements die Gruppenordnung teilt, brauchen wir nur die Teiler durchzuprobieren:

Satz 5.25 Es sei $|G| = p_1^{k_1} \cdots p_r^{k_r}$ die Primzahldarstellung der Gruppenordnung. Dann gilt

$$\operatorname{ord}(a) = p_1^{l_1} \cdots p_r^{l_r}$$

wobei $0 \le l_j \le k_j$ die kleinste Zahl mit $a^{|G| p_j^{l_j - k_j}} = n$ ist.

Insbesondere ist a genau dann ein Generator von G, wenn

$$a^{|G|/p_j} \neq n$$

für alle $1 \le j \le r$.

Da $\text{ord}(a)$ ein Teiler von $|G|$ kann es so wie oben dargestellt werden. Erhöhen wir eines der l_j so erhalten wir ein Vielfaches der Ordnung und es gilt weiterhin $a^{|G|}p_j^{l_j+1-k_j} = n$. Erniedrigen wir eines der l_j (mit $l_j > 0$), so gilt $a^{|G|}p_j^{l_j-1-k_j} \neq n$, da die Ordnung ja als kleinste natürliche Zahl mit dieser Eigenschaft definiert wurde.

Damit haben wir eine effizienten Algorithmus zur Berechnung von $\text{ord}(a)$, wenn die Faktorisierung von $|G|$ bekannt ist. In bestimmten Situtationen gilt auch die Umkehrung, wir werden darauf in Abschnitt 5.5.6 zurückkommen.

Folgende zyklische Gruppen spielen in der Kryptographie eine wichtige Rolle:

Satz 5.26 Ist $\mathbb{K}$ ein Körper, so ist $(\mathbb{K}^*, \cdot)$ zyklisch.

Es sei $|\mathbb{K}^*| = m = p_1^{k_1} \cdots p_r^{k_r}$ die Primfaktorzerlegung der Gruppenordnung m. Betrachten wir das Polynom $P(a) = a^{m/p_j} - 1$. Da P höchstens m/p_j Nullstellen hat, muss es ein $b_j \in \mathbb{K}^*$ mit $b_j^{m/p_j} \neq 1$ geben. Dann hat $a_j = b_j^{m/p_j^{k_j}}$ die Ordnung $p_j^{k_j}$, denn $a_j^{p_j^{k_j}} = b_j^m = 1$ und die Ordnung kann nicht kleiner sein, da $a_j^{p_j^{k_j-1}} = b_j^{m/p_j} \neq 1$. Nach Satz 5.18 hat also $a = a_1 \cdots a_r$ die Ordnung m.

Für jede Primzahl p ist also $(\mathbb{Z}_p^*, \cdot)$ zyklisch.

Falls m keine Primzahl ist, so kann $(\mathbb{Z}_m^*, \cdot)$ zyklisch sein oder nicht, wie wir in Beispiel 5.9 gesehen haben. Damit $\mathbb{Z}_m^*$ zyklisch ist muss es ein Element der Ordnung $\varphi(m)$ haben. Die maximale Ordnung die ein Element in $\mathbb{Z}_m^*$ hat wird als **Carmichael-Funktion** $\lambda(m)$ bezeichnet.

Beispiel 5.27 Aus Beispiel 5.9 sehen wir
a) $\lambda(7) = 6 = \varphi(7)$ b) $\lambda(8) = 2 < 4 = \varphi(8)$ c) $\lambda(10) = 4 = \varphi(10)$
Die zyklischen Gruppen sind genau die mit $\lambda(m) = \varphi(m)$.

Insbesondere muss jede Ordnung eines Elements aus $\mathbb{Z}_m^*$ ein Teiler von $\lambda(m)$ sein (sonst würde Satz 5.18 ein Element mit größerer Ordnung liefern). Äquivalent können wir also $\lambda(m)$ als kleinste Zahl definieren, für die der Satz von Euler gilt:

Satz 5.28 Es sei $\lambda(m)$ die Carmichael-Funktion, dann gilt

$$a^{\lambda(m)} = 1 \pmod{m}$$

für alle $a \in \mathbb{Z}_m^*$ gilt und $\lambda(m)$ ist die kleinste Zahl mit dieser Eigenschaft.

In jedem Fall muss $\lambda(m)$ ein Teiler von $\varphi(m)$ sein und im Fall einer Primzahl gilt laut Satz 5.26

$$\lambda(p) = \varphi(p) = p - 1.$$

Satz 5.29 Sind m und n teilerfremd so gilt

$$\lambda(m \cdot n) = \mathrm{kgV}(\lambda(m), \lambda(n)).$$

In diesem Fall sagt uns der Chinesische Restsatz, dass $\mathbb{Z}_{mn}$ isomorph zu $\mathbb{Z}_m \times \mathbb{Z}_n$ ist. Nun gilt aber $(a,b)^{\mathrm{kgV}(\lambda(m),\lambda(n))} = (a^{\mathrm{kgV}(\lambda(m),\lambda(n))}, b^{\mathrm{kgV}(\lambda(m),\lambda(n))}) = (1,1)$ und somit $\lambda(m \cdot n) \leq \mathrm{kgV}(\lambda(m), \lambda(n))$. Umgekehrt können wir ein Element $a \in \mathbb{Z}_m$ mit $\mathrm{ord}(a) = \lambda(m)$ und ein Element $b \in \mathbb{Z}_n$ mit $\mathrm{ord}(a) = \lambda(n)$ wählen. Dann gilt $\mathrm{ord}((a,b)) = \mathrm{kgV}(\lambda(m), \lambda(n))$, also $\lambda(m \cdot n) \geq \mathrm{kgV}(\lambda(m), \lambda(n))$.

Damit kann man die Berechnung auf Primzahlpotenzen reduzieren. Für das Produkt zweier Primzahlen folgt damit

$$\lambda(p \cdot q) = \mathrm{kgV}(p-1, q-1).$$

Allgemein kann man

$$\lambda(p^k) = \begin{cases} \frac{\varphi(p^k)}{2}, & p = 2 \text{ und } k \geq 3, \\ \varphi(p^k), & \text{sonst} \end{cases}$$

zeigen. Daraus folgt dann, dass $\mathbb{Z}_m$ genau dann zyklisch ist, wenn $m = 1, 2, 4$, p^k oder $2p^k$ mit $p \geq 3$ prim ist. Wir werden das aber nicht benötigen.

5.3 Sicherheit des DH-Schlüsseltausch und des DLP

Wir beginnen mit der bereits erwähnten Tatsache, dass man den DH-Schlüsseltausch im Prinzip auf beliebigen Gruppen durchführen kann und das diese Tatsache in der Kryptographie eine wichtige Rolle spielt. Deshalb betrachten wir diesen allgemeineren Fall.

Beim DH-Schlüsseltausch erfährt die Angreiferin Eve die beiden Werte $\alpha = g^a$ und $\beta = g^b$ und muss daraus das geteilte Geheimnis $k = g^{ab}$ berechnen. Das wird als **DH-Problem** bezeichnet. Die naheliegendste Strategie ist es, das DLP $\alpha = g^a$ (oder $\beta = g^b$) zu lösen. Es ist aber nicht bekannt ob es nicht auch einen anderen Weg gibt, bzw. etwas allgemeiner, ob eine effektive Lösung des DH-Problems auch eine effektive Lösung des DLP ermöglicht.

Ein alternatives Angriffszenario wäre es z.B., wenn man die möglichen Werte für k andhand der Eigenschaften von α und β einschränken kann, damit ein Durchsuchen der verbleibenden Möglichkeiten praktisch durchführbar wird. Eine mögliche Beobachtung ist z.B. die Tatsache, dass k genau dann eine Quadratzahl (also von der Form g^j mit einer geraden Potenz j) ist, wenn das für α oder β zutrifft. In $\mathbb{Z}_p^*$ kann man das effektiv mithilfe des Legendre-Symbols (siehe Abschnitt 7.2) feststellen und damit die Anzahl der möglichen k um die Hälfte reduzieren. Für einen praktischen Angriff reicht das aber nicht.

Nach diesen Vorbemerkungen können wir uns nun etwas näher mit der Sicherheit von Algorithmen, die auf dem DLP

$$g^a = \alpha$$

basieren, auseinandersetzen. Zunächst ist klar, dass ein naiver Brute-Force Angriff maximal $\mathrm{ord}(a)$ Möglichkeiten durchprobieren muss. Also sollte $\mathrm{ord}(a)$ möglichst groß sein.

Diese Bedingung würde für einen Generator (falls G zyklisch ist) sprechen. Allerdings ergibt sich beim DHKE ein Problem, wenn $\alpha = g^a$ eine kleine Ordnung hat, dann gibt es nur noch $r = \mathrm{ord}(\alpha)$ Möglichkeiten für das gemeinsame Geheimnis k, die Eve durchprobieren muss. Nach Satz 5.17 gilt

$$\mathrm{ord}(\alpha) = \frac{\mathrm{ord}(g)}{\mathrm{ggT}(\mathrm{ord}(g), a)}$$

und im ungünstigsten Fall wählt Alice $a = \frac{p-1}{2}$ und dann ist $\alpha^2 = 1$, erzeugt also nur eine sehr kleine Untergruppe ($\{1\}$ oder $\{1, p-1\}$), weswegen dann $k = \alpha^b$ leicht für Mallory zu erraten ist. Man sollte also Teiler von $\mathrm{ord}(g)$ (bzw. Teiler von $p-1$, falls die Ordnung nicht bekannt ist) für die Wahl von a ausschließen. Das ist aber umständlich und kann vermieden werden, indem man ein Element wählt, dass eine große Ordnung die prim ist besitzt. Dazu muss aber $p-1$ einen großen Primteiler besitzen. Das kann wie folgt erreicht werden:

- Wähle eine Primzahl q in der gewünschten Größenordnung (das wird der Primteiler).
- Betrachte $p = 2rq+1$ und erhöhe $r \geq 1$ so lange um eins bis p ebenfalls prim ist.
- Wähle ein $h \in \mathbb{Z}_p^*$ und teste ob $h^{2r} = 1 \pmod{p}$. In diesem (unwahrscheinlichen) Fall wähle ein neues h und wiederhole den Test.
- Dann hat $g = h^{2r} \pmod{p}$ die Ordnung q.

Dass $p-1$ den Primteiler q besitzt folgt unmittelbar aus der Konstruktion. Die Bedingung $h^{2r} \neq 1 \pmod{p}$ stellt sicher, dass $\mathrm{ord}(h)$ kein Teiler von $2r$ ist (also h nicht in der Untergruppe H_{2r} aus Satz 5.20). Also muss $\mathrm{ord}(h^{2r}) = q$ sein. Warum der Algorithmus immer ein Primzahl findet und wie man feststellt, ob eine Zahl prim ist, werden wir in Abschnitt 5.6 besprechen.

Der in diesem Sinn beste Fall tritt ein, wenn $q = \frac{p-1}{2}$ auch prim ist. Solche Primzahlen werden als **Safe-Prime** bezeichnet und q wird dann als **Sophie-Germain-Primzahl** bezeichnet (nach der französischen Mathematikerin Sophie Germain, 1776–1831).

Beispiel 5.30 Die ersten Sophie-Germain-Primzahlen sind

$$2, 3, 5, 11, 23, 29, 41, 53, 83, 89, \ldots$$

und die zugörigen Safe-Prime sind

$$5, 7, 11, 23, 47, 59, 83, 107, 167, 179, \ldots.$$

Satz 5.31 Ist $p = 2q+1$ eine Safe Prime, so hat $a \in \mathbb{Z}_p^* \setminus \{\pm 1\}$ entweder Ordnung q falls $a^q = 1$ oder $2q$ falls $a^q = -1$ gilt.

Als Ordnung kommen nur die Teiler 1, 2, q, $2q$ der Gruppenordnung $|\mathbb{Z}_p^*| = 2q$ in Frage. Ist $a^2 = 1$, so hat a die Ordnung 1 oder 2. Da die Gleichung $a^2 - 1 = (a+1)(a-1) = 0$ im Körper $\mathbb{Z}_p$ nur die Lösungen $a = \pm 1$ hat, gibt es also nur die beiden Möglichkeiten $a = 1$ mit $\mathrm{ord}(1) = 1$ und $a = -1 = p-1$ mit $\mathrm{ord}(p-1) = 2$. Ist $a^q = 1$ mit $a \neq 1$, so gilt $\mathrm{ord}(a) = q$ und anderenfalls, also $a^q \neq 1$, $\mathrm{ord}(a) = 2q$ und a ist ein Generator. Ein Element $1 < a < p-1$ ist also genau dann ein Generator, wenn $a^q = -1$.

In der Praxis findet man solche Zahlen wie folgt:

- Wähle sukzessive Primzahlen q in der gewünschten Größenordnung.
- Teste ob $p = 2q + 1$ auch eine Primzahl ist. Wenn ja, fertig. Wenn nein, wiederhole den letzten Schritt.

Da Primzahltests aufwendig sind (siehe Abschnitt 5.6), kann man zur Beschleunigung vorweg notwendige Bedingungen abfragen (siehe z.B. Aufgabe 14). Es ist allerdings nicht bekannt, ob es unendlich viele Sophie-Germain-Primzahlen gibt. Es könnte also sein, dass der Algorithmus erfolglos ist.

In der Praxis gibt es noch weitere Faktoren, die berücksichtigt werden sollen (vgl. [Adr+15]) und man verwendet am besten standardisierte Werte [RFC3526].

Für eine Safe Prime $p = 2q + 1$ gibt es auch eine zugehörige Hashfunktion deren Kollissionsresistenz auf dem DLP beruht. Die **Chaum-van Heijst-Pfitzmann-Hashfunktion**

$$H : \mathbb{Z}_q \times \mathbb{Z}_q \to \mathbb{Z}_p^*, \qquad (x_1, x_2) \mapsto g_1^{x_1} g_2^{x_2} \pmod{p},$$

wobei g_1 und g_2 zwei Generatoren sind, für die $\log_{g_1}(g_2)$ nicht bekannt ist. Dann kann aus einer Kollision $\log_{g_1}(g_2)$ berechnet werden (vgl. [Wät08]). Allerdings folgt aus den Potenzgesetzten, dass die Hashfunktion algebraische Beziehungen in der Nachricht überträgt und somit für kryptographische Anwendungen eher ungeeignet ist.

Brute Force ist nicht der einzig mögliche Angriff auf das DLP. Es gibt mathematische Algorithmen, die deutlich effizienter sind (und die auch miteinander kombiniert werden können). Wir besprechen die wichtigsten in den folgenden Unterabschnitten. Man beachte, dass die ersten drei Verfahren keine speziellen Eigenschaften der Gruppe ausnutzen und somit auf jede zyklische Gruppe G ($\mathbb{Z}_p^*$,

Punktgruppe einer elliptischen Kurve, ...) anwendbar sind. Alle haben (bei entsprechend gewählten Parametern der Gruppe) einen Aufwand von $O(\sqrt{|G|})$ und das erklärt auch die empfohlenen Schlüssellängen für elliptische Kurven (siehe Tabelle 2.2 in Studienbrief 2, letzte Spalte; ECDH = „Elliptic Curve Diffie Hellman“). Die letzte (und effektivste) Methode funktioniert nur in $\mathbb{Z}_p^*$ und bedeutet somit eine weitere Reduktion der Sicherheit in diesem Fall; das erklärt die deutlich größeren Schlüssellängen in Spalte 3 in Tabelle 2.2.

Ein Resultat von Shoup [Sho97b] besagt, dass, alleine unter Verwendung der Gruppenoperationen, $O(\sqrt{|G|})$ nicht verbessert werden kann. Alle Verfahren die asymptotisch schneller sind, müssen also spezielle Eigenschaften der Gruppe verwenden.

Für Implementation dieser Verfahren am Computer und weitere mathematische Details sei auf [For15] verwiesen.

5.3.1 Baby Step-Giant Step

Wir beginnen mit dem **Baby Step-Giant Step-Algorithmus** von Shanks (Daniel Shanks (1917–1996), US-amerikanischer Mathematiker). Dazu sei G eine Gruppe und $a \in G$ ein Element dieser Gruppe mit $\mathrm{ord}(a) > 2$. Für gegebenes $b \in G$ ist $x \in \mathbb{N}$ mit $a^x = b$ gesucht.

Die Idee ist nun wie folgt: Wir wählen $m = \lceil \sqrt{\mathrm{ord}(a)} \rceil$ und zerlegen $x = qm + r$ mit $0 \leq r < m$. Es gilt $q = \frac{x-r}{m} < \frac{\mathrm{ord}(a)}{m} \leq m$ (da $m \geq \sqrt{\mathrm{ord}(a)}$). Damit können wir $a^x = b$ als $a^r = b \circ a^{-qm}$ mit $0 \leq r < m$ und $0 \leq q < m$ schreiben. Rechnen wir also a^k für $0 \leq k < m$ und a^{-lm} für $0 \leq l < m$ aus, so muss es bei $k = r$ und $l = q$ eine Übereinstimmung geben, aus der wir x ermitteln können. Das führt auf folgenden Algorithmus:

- Setzte $m = \lceil \sqrt{\mathrm{ord}(a)} \rceil$ und berechne $c = a^{-m}$.
- Berechne die beiden Listen

$$n, a, a^2, \ldots, a^{m-1}, \qquad b, b \circ c, b \circ c^2, \ldots, b \circ c^{m-1}.$$

- Finde zwei gleiche Elemente in den Listen: $a^k = b \circ c^l = b \circ a^{-lm}$.
- Dann ist $x = k + ml$ der gesuchte Wert.

Die Multiplikation mit a ist der „Baby Step“ und die Multiplikation mit a^{-m} der „Giant Step“. Daher der Name.

Beispiel 5.32 Berechnen Sie $\log_3(18)$ in $\mathbb{Z}_{23}^*$ mit dem Baby Step-Giant Step-Algorithmus.

Lösung zu 5.32 Wir haben also $a = 3$, $b = 18$, $p = 23$ und $q = \frac{p-1}{2} = 11$. Wegen $a^q = 1$ gilt $\text{ord}(a) = q$ und $m = \lceil\sqrt{11}\rceil = 4$ bzw. $a^{-m} = 2$. Die beiden Listen sind

$$1, 3, 9, 4, \qquad 18, 13, 3, 6.$$

Da $3 = a^1$ in der ersten Liste gleich $3 = b(a^{-m})^2$ in der zweiten Liste ist, folgt $\log_3(18) = 1 + 2m = 9$. ■

Aufstellen der Listen und die Suche benötigt $O(m\log(m))$ Operationen. Falls man die Ordnung nicht kennt, so kann man natürlich mit $m = \lceil\sqrt{|G|}\rceil$ arbeiten. Erstellt man die Listen parallel, so kann man abbrechen sobald eine Übereinstimmung gefunden wurde und sogesehen wird m nur benötigt um zu wissen, wann man die Suche aufgeben kann (falls $b \notin \langle a\rangle$ und das Problem somit unlösbar ist).

Insgesamt ist der Algorithmus also von der Ordnung $O(\sqrt{|m|}\log(|m|))$ und benötigt $O(\sqrt{|m|})$ Speicherplatz.

Die Wurzel bedeutet, dass die Anzahl der (Binär-)Stellen zwar halbiert wird, das ändert aber nichts daran, dass die Laufzeit des Algorithmus exponentiell mit der Anzahl der Stellen steigt: $O(\sqrt{m}) = O(2^{\frac{1}{2}\log_2(m)})$. Es handelt sich also um einen Algorithmus mit exponentieller Laufzeit. Im Gegensatz zur Exponentiation, die mit Square-and-Multiply polynomial mit der Anzahl der Stellen wächst: $O(\log_2(m)^3)$.

5.3.2 Pohlig–Hellman Reduktion

Die Pohlig–Hellman Reduktion (nach Martin Hellman und seinem Studenten Stephen Pohlig) zerlegt das DLP in kleinere Probleme anhand der Faktorisierung der Ordnung.

Gehen wir von

$$a^x = b$$

aus und nehmen wir die m'te Potenz, so gilt

$$(a^m)^x = b^m.$$

Wegen $\text{ord}(a^m) = \frac{\text{ord}(a)}{\text{ggT}(\text{ord}(a),m)}$ lebt das neue DLP im allgemeinen in einer kleineren Untergruppe und ist dementsprechend leichter zu lösen. Können wir insbesondere die Ordnung in zwei Faktoren $\text{ord}(a) = m = m_1m_2$ zerlegen, so führt uns das auf zwei kleinere Probleme

$$(a_1)^{x_1} = b_1, \qquad (a_2)^{x_2} = b_2$$

mit $a_1 = a^{m_2}$, $b_1 = b^{m_2}$ und $a_2 = a^{m_1}$, $b_2 = b^{m_1}$. Nach Konstruktion gilt $x_j = x \bmod m_j$ und das schreit nach dem Chinesischen Restsatz: Sind m_1 und m_2 teilerfremd, so können wir x mit dem Chinesischen Restsatz aus

$$x \bmod m_1 = x_1, \qquad x \bmod m_2 = x_2$$

bestimmen.

Beispiel 5.33 Berechnen Sie $\log_5(18)$ in $\mathbb{Z}_{23}^*$ mit der Pohlig-Hellman Reduktion.

Lösung zu 5.33 Wir haben also $a = 5$, $b = 18$, $p = 23$ und $q = \frac{p-1}{2} = 11$. Wegen $a^q = -1$ gilt $\text{ord}(a) = 2q$. Das führt uns auf die kleineren Probleme $a_1 = a^q = 22$, $b_1 = b^q = 1$ und $a_2 = a^2 = 2$, $b_2 = b^2 = 2$. Die Lösungen sind offensichtlich $x_1 = 0$ und $x_2 = 1$ und mit dem Chinesischen Restsatz folgt $\log_5(18) = 12$. ■

Natürlich können wir die Ordnung weiter zerlegen. Ist $\text{ord}(a) = m = p_1^{k_1} \cdots p_r^{k_r}$ die Primfaktorzerlegung der Ordnung von a, so reicht es die kleineren Probleme

$$a_j^{x_j} = b_j, \qquad a_j = a^{m/p_j^{k_j}},\ b_j = b^{m/p_j^{k_j}}$$

zu lösen und man kann dann $\log_a(b)$ mithilfe des Chinesischen Restsatzes aus $\log_a(b) = x_j \pmod{p_j^{k_j}}$ bestimmen.

In einem weitern Schritt kann man dann noch rekursiv das Bestimmen des diskreten Logarithmus modulo einer Primzahlpotenz p^k auf das Bestimmen des diskreten Logarithmus modulo der Primzahl p zurückführen. Um das zu sehen betrachten wir $G = \langle g \rangle$ mit $|G| = p^k$. Dann wissen wir (Satz 5.20), dass es zu jedem Teiler p^j genau eine Untergruppe $H_j = \langle g_j \rangle$, mit $g_j = g^{p^{k-j}}$ und $\text{ord}(g_j) = p^j$, gibt. Diese bilden eine aufsteigende Folge von Untergruppen

$$\{n\} = H_0 \subset H_1 \subset H_2 \subset \cdots \subset H_{k-1} \subset H_k = G.$$

Wollen wir $g^x = b$ lösen, so gehen wir wie folgt vor: Wir definieren $b_j = b^{p^{k-j}} \in H_j$ und beginnen mit $x_1 = \log_{g_1}(b_1)$. Angenommen wir haben nun x_j mit

$$b_j = g_j^{x_j}$$

gefunden. Dann gilt $(b_{j+1} g_{j+1}^{-x_j})^p = b_{j+1}^p (g_{j+1}^p)^{-x_j} = b_j g_j^{-x_j} = n$, also $b_{j+1} g_{j+1}^{-x_j} \in H_1$ und wir können

$$y_j = \log_{g_1}(b_{j+1} g_{j+1}^{-x_j})$$

setzten. Damit folgt

$$b_{j+1} = g_{j+1}^{x_j} g_1^{y_j} = g_{j+1}^{x_{j+1}}$$

mit $x_{j+1} = x_j + p^j y_j$. Wegen $b_k = b$ und $g_k = g$ ist $x_k = x$ die gesuchte Lösung.

Beispiel 5.34 Berechnen Sie $\log_3(2)$ in $\mathbb{Z}_5^*$ mit der Pohlig-Hellman Reduktion.

Lösung zu 5.34 Es ist $\varphi(5) = 2^2$ und da 5 prim ist, ist $g = 3$ ein Generator. Es gilt $g_1 = 3^{2^{2-1}} = 4$, $b_1 = 2^{2^{2-1}} = 4$ und somit $x_1 = 1$. Weiters folgt $g_2 = g = 3$ und $b_2 = b = 2$ also $y_1 = \log_{g_1}(b_2 g_2^{-x_1}) = \log_4(2 \cdot 3^{-1}) = \log_4(4) = 1$. Daraus folgt $\log_3(2) = x_2 = x_1 + 2y_1 = 3$. ■

Das ist die **Pohlig–Hellman Reduktion** und damit reduziert sich der Aufwand auf $O(\sqrt{p_r})$, wobei p_r der größte Primfaktor von $|G|$ ist. Somit muss $|G|$ möglichst große Primfaktoren besitzen. Im Fall von $\mathbb{Z}_p^*$ ist also wieder eine Safe Prime eine gute Wahl.

Um diese Methode anwenden zu können, muss man die Ordnung und ihre Primfaktorzerlegung kennen. In der Praxis könnte man hier unter Umständen schon scheitern. Allerdings würde auch schon ein Teil der Primfaktoren reichen, da es für den Chinesischen Restsatz ja ausreicht, wenn die Module teilerfremd sind.

5.3.3 Pollard Rho

Die Idee von **Pollard's Rho Methode** ist es, dass man anstelle eines bestimmten Wertes, eine Kollision sucht, da letztere nach dem Geburtstagsparadoxon mit wesentlich höherer Wahrscheinlichkeit auftreten. Dieses (und eine Reihe weiterer) Verfahren stammen vom britischen Mathematiker John M. Pollard (*1941). Wir erklären das Prinzip etwas allgeminer, da sich die gleiche Idee auch in anderen Situationen verwenden läßt (z.B. zum Faktorisieren, worauf wir später noch zurückkommen werden).

Gegeben ist also eine Menge M und wir suchen eine Kollision in dem Sinne, dass wir zwei Elemente mit der gleichen Eigenschaft suchen (also z.B. zwei Personen in einer gegeben Menge, die am gleichen Tag Geburtstag haben). Da echte Zufallsfolgen in der Praxis nur aufwendig zu realisieren sind, generieren wir eine Pseudozufallsfolge indem wir eine (noch passend zu wählende) Funktion $f : M \to M$ rekursiv auf einen Startwert $x_0 \in M$ anwenden, also

$$x_{j+1} = f(x_j).$$

Das typische Bild, das sich dabei ergibt ist in Abbildung 5.2 veranschaulicht. Ab irgendeinem Index $j = n_0 + k$ kommt es zum ersten Mal zu einer Kollision $x_{n_0} = x_{n_0+k}$. Ab diesem Index ist die Folge zum Punkt x_{n_0} zurückgekehrt und wiederholt sich von nun an periodisch mit der Periode k:

$$x_j = x_{j+k}, \qquad j \geq n_0.$$

Das sieht man sofort mit Induktion. Der Induktionsanfang ist die erste Kollision $j = n_0$ und durch Anwenden von f auf die Gleichung für j folgt die Gleichung für $j + 1$: $x_{j+1} = f(x_j) = f(x_{j+k}) = f_{j+1+k}$.

Die (minimal gewählten) Werte n_0 bzw. k nennt man die Vorperiode bzw. Periode von x_0. In Abbildung 5.2 gilt $x_2 = x_{11}$ und somit ist die Vorperiode $n_0 = 2$ und die Periode $k = 9$.

Verbindet man die Punkte in Abbildung 5.2 der Reihe nach („Malen nach Zahlen"), so erhält man mit etwas Phantasie den griechischen Buchstaben ρ, was den Namen erklärt.

Nun muss man aber zum Auffinden der ersten Kollision alle Werte der Folge x_j abspeichern und jeden neuen Wert mit den bisherigen vergleichen. Das ist sehr

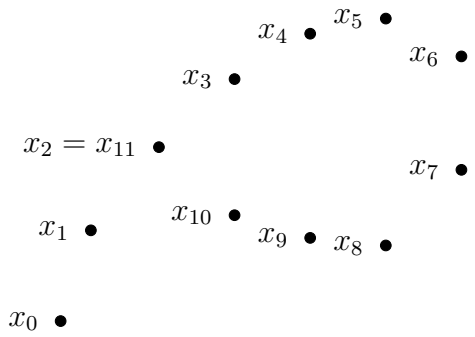


Abbildung 5.2: Pollard'sche ρ Methode: Die iterierten Werte eine Funktion führen nach einer Vorperiode in einen periodischen Zykel.

aufwendig und kann den Vorteil der Idee zunichte machen. Deshalb verwendet man folgenden Trick: Man berechnet

$$x_{j+1} = f(x_j), \qquad y_{j+1} = f(f(y_j))$$

mit dem gleichen Startwert $x_0 = y_0$, sodass $y_j = x_{2j}$ gilt, und wartet bis zum ersten Mal $x_{j_0} = y_{j_0}$ auftritt. Nach Konstruktion gilt $x_{j_0} = x_{2j_0} = x_{j_0+j_0}$ woraus wir durch Vergleich mit vorher ablesen können, dass j_0 eine Vielfaches der (minimalen) Periode k sein muss (da aus $x_{j_0} = x_{2j_0}$ durch wiederholtes Anwenden von f auch $x_j = x_{j+j_0}$ für $j \geq j_0$ folgt).

Wäre j_0 kein Vielfaches von k, so könnten wir es zuerst modulo k reduzieren und hätten dann eine Periode die kleiner als k ist, im Widerspruch zur Minimalität von k.

Ist insbesondere $n_0 \leq k$, so passiert $y_j = x_{2j}$ zum ersten Mal für $j_0 = k$ und wir erhalten die gewünschte Kollision genauso schnell, ohne aber alle Ergebnisse zwischenspeichern zu müssen. Wir hoffen also, dass dieser Fall eintritt.

Soweit die prinzipielle Idee, wenngleich das Verfahren in der Praxis nicht immer genau so angewandt wird, da wir die Kollision manchmal nicht in den x_j selbst, sondern nur in den Eigenschaften der x_j suchen (unsere Pseudozufallsfolge muss also nicht wieder auf die gleiche Person treffen, sondern nur auf eine Person mit dem gleichen Geburtstag).

Umgesetzt wird diese Idee beim diskreten Logarithmus wie folgt: Wir teilen unsere Gruppe in drei ungefähr gleich große (disjunkte) Teile $G = G_0 \cup G_1 \cup G_2$ und definieren

$$f(x) = \begin{cases} x \circ a, & x \in G_0, \\ x^2, & x \in G_1, \\ x \circ b, & x \in G_2 \end{cases}$$

mit Startwert

$$x_0 = a^{j_0} \circ b$$

mit einer Zufallszahl j_0. Dann gilt,

$$x_j = a^{\alpha_j} \circ b^{\beta_j},$$

wobei die Zahlen α_j, β_j bei der rekursiven Auswertung leicht mitbestimmt werden können

$$(\alpha_{j+1}, \beta_{j+1}) = \begin{cases} (\alpha_j + 1, \beta_j), & x_j \in G_0, \\ (2\alpha_j, 2\beta_j), & x_j \in G_1, \\ (\alpha_j, \beta_j + 1), & x_j \in G_2. \end{cases}$$

Aus einer Kollision $x_j = x_k$ erhält man die Gleichung

$$a^{\alpha_j - \alpha_k} = b^{\beta_k - \beta_j}$$

und somit

$$\alpha_j - \alpha_k = x(\beta_k - \beta_j) \pmod{\operatorname{ord}(a)}$$

für den gesuchten diskreten Logarithmus x. Ist $\beta_k - \beta_j$ teilerfremd zu $\operatorname{ord}(a)$, so können wir mit dem multiplikativen Inversen λ in $\mathbb{Z}_{\operatorname{ord}(a)}$ (also $(\beta_k - \beta_j)\lambda = 1 \pmod{\operatorname{ord}(a)}$) multiplizieren um die Lösung

$$b = a^{(\alpha_j - \alpha_k)\lambda}$$

unseres DLP zu erhalten. Ist $\beta_k - \beta_j$ nicht teilerfremd zu $\operatorname{ord}(a)$, so gibt es zwei Möglichkeiten: Ist $\beta_k - \beta_j = 0 \pmod{\operatorname{ord}(a)}$, haben wir Pech gehabt und müssen mit einer neuen Zufallszahl j_0 von vorne beginnen. Ansonsten kann man beide Seiten der Gleichung durch $t = \operatorname{ggT}(\beta_k - \beta_j, \operatorname{ord}(a))$ teilen und danach lösen. Das liefet $x_0 = x \pmod{\operatorname{ord}(a)/t}$ und man kann die verbleibenden Möglichkeiten $x_0 + j \operatorname{ord}(a)/t$, $0 \leq j < t$ durchprobieren falls t nicht zu groß ist (ansonsten versucht man eine neue Zufallszahl j_0).

Kennt man die Ordnung $\operatorname{ord}(a)$ nicht, so kann man sie auch durch ein Vielfaches, z.B. $|G|$, ersetzten. Dadurch wird das Verfahren aber öfter scheitern und somit weniger effektiv sein.

Im Fall $G = \mathbb{Z}_p$ kann für die Aufteilung z.B. der Rest von x modulo 3 genommen werden.

Beispiel 5.35 Berechnen Sie $\log_3(16)$ in $\mathbb{Z}_{23}^*$ mit der Pollard ρ Methode.

Lösung zu 5.35 Wir haben also $a = 3$, $b = 16$, $p = 23$ und $q = \frac{p-1}{2} = 11$. Wegen $a^q = 1$ gilt $\operatorname{ord}(a) = q$. Wir wählen eine Startpotenz, z.B. $j_0 = 2$. Dann sind die zugehörigen Startwerte

$$x_0 = y_0 = a^2 \cdot b = 6.$$

Nun folgt $x_1 = f(x_0) = 18 = a^3 \cdot b^1$ da $x_0 \bmod 3 = 0$ und $y_1 = f(x_1) = 8 = a^4 \cdot b^1$ da $x_1 \bmod 3 = 0$.

In der nächsten Runde kennen wir schon $x_2 = y_1$ und es gilt $y_2 = f(f(y_1)) = 8 = a^8 \cdot b^4$. Das ist schon die gesuchte Kollision $a^4 \cdot b^1 = a^8 \cdot b^4$ und somit $b = a^6$. Die beiden möglichen Werte für $\log_3(18)$ sind also 6 und $6 + q = 17$. ■

Als Anzahl der notwendigen Iterationen erwarten wir $O(\sqrt{m})$ mit $m = \text{ord}(a)$, aber die tatsächliche Anzahl kann davon abweichen und hängt insbesondere vom Startwert x_0 ab. Es ist also sinnvoll eine Abbruchbedingung einzubauen und nach (z.B.) $10\sqrt{m}$ Iterationen mit einem neuen Startwert zu beginnen.

5.3.4 Index Calculus

Wir suchen $\log_a(b)$ in $\mathbb{Z}_p$. Dazu wählen wir irgendein x und faktorisieren

$$b \cdot a^x = p_1^{k_1} \cdots p_r^{k_r}.$$

Dann gilt

$$\log_a(b) + x = \sum_{i=1}^{r} k_i \log_a(p_i).$$

Das ist eine lineare Gleichung für die unbekannten Logarithmen $\log_a(b)$, $\log_a(p_1)$, $\ldots$, $\log_a(p_n)$ im Ring $\mathbb{Z}_m$ mit $m = \text{ord}(a)$. Die Idee ist es nun genügend viele solcher Gleichungen zu finden, damit man das zugehörige Gleichungssystem nach den unbekannten Logarithmen auflösen kann. Dieser Idee stehen allerdings zwei Probleme im Weg: Erstens ist das Faktorisieren schwer und zweitens kommen mit jeder neuen Zahl neue Primzahlen dazu und dann brauchen wir wieder neue Gleichungen.

Die Lösung für beide Probleme ist nun folgende: Wir wählen eine **Faktorbasis** $B = \{p_1, \ldots, p_r\}$, also eine feste Menge von Primzahlen (z.B. alle Primzahlen die kleiner als eine vorgegebene Schranke sind). Eine Zahl die sich mit den Primzahlen aus B faktorisieren läßt (z.B. durch Probedivision) nennt man B**-glatt**.

Der **Index Calculus** ist nun wie folgt: Wähle zufällig $x_j \in \mathbb{Z}_m$. Ist $b \cdot a^{x_j} = p_1^{k_{1,j}} \cdots p_r^{k_{r,j}}$ B-glatt, dann notieren wir die zugehörige Gleichung

$$-\log_a(b) + \sum_{i=1}^{r} k_{i,j} \log_a(p_i) = x_j.$$

Dieser Teil kann parallel ausgeführt werden. Die gefundenen Gleichungen werden in eine Matrix

$$\begin{pmatrix} -1 & k_{1,1} & k_{2,1} & \cdots & k_{r,1} & x_1 \\ -1 & k_{1,2} & k_{2,2} & \cdots & k_{r,2} & x_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \end{pmatrix}$$

geschrieben die in $\mathbb{Z}_m$ auf reduzierte Zeilenstufenform (Gauß-Algorithmus) gebracht wird. Das kann laufend geschehen und sobald man das Gleichungssystem nach $\log_a(b)$ lösen kann ist man fertig. Im Prinzip muss m auch nicht prim sein, aber das Lösen des Gleichungssytems wird dann aufwendiger, da nicht mehr jedes von Null verscheidene Element ein multiplikatives Inverses hat.

Der Name Index Calculus kommt daher, dass der diskrete Logarithmus füher als Index bezeichnet wurde.

Beispiel 5.36 Die Faktorbasis sei $B = \{2, 3\}$ und $p = 23$. Für welche Zahlen $j = 1, 2, 3, 4$ ist $b \cdot a^j \bmod p$ B-glatt? Berechnen Sie $\log_3(18)$ in $\mathbb{Z}_{23}^*$ mittels Index Calulus.

Lösung zu 5.36 Wir haben also $a = 3$, $b = 18$ und $p = 23$. Es gilt

$$b \cdot a^1 = 8 = 2^8 3^0, \quad b \cdot a^2 = 1 = 2^0 3^0, \quad b \cdot a^3 = 3 = 2^0 3^1, \quad b \cdot a^4 = 9 = 2^0 3^2$$

und somit sind alle Zahlen B-glatt. (Die erste nicht B-glatte Zahl wäre $b \cdot a^7 = 13$.) Die zweite Zahl $j = 2$ ist schon der Jackpot, denn sie besagt ja $\log_8(18) = -2 = 20 \in \mathbb{Z}_{p-1}$ (bzw. $\log_8(18) = 9 \in \mathbb{Z}_{\mathrm{ord}(a)}$, da $\mathrm{ord}(a) = 11$). Aber auch $j = 3, 4$ liefern ein Gleichungssystem, das wir nach $\log_8(18)$ auflösen können. ■

Ob eine Zahl B-glatt ist stellt man mittels Probedivision fest. Das ist rechenintensiv, insbesondere da die meisten Divisionen nicht aufgehen. Um diese Fehlversuche zu vermeiden, siebt man die Zahlen zuvor aus. Wir besprechen das in Abschnitt 5.5.5 etwas genauer. Die Laufzeit des Algorithmus ist subexponentiell $O(\exp(2\sqrt{\log(p)\log(\log(p))}))$. Man beachte, dass die Laufzeit durch p und nicht durch $m = \mathrm{ord}(a)$ bestimmt ist.

Ein Satz von Canfield-Erdös-Pomerance besagt, dass die Wahrscheinlichkeit, dass eine Zahl n nur Primfaktoren $\leq n^{1/u}$ besitzt, ungefähr $1/u^u$ ist. Für $u = 4$ erhalten wir, dass ca. jede 256te Primzahl mit l Stellen nur Primfaktoren mit maximal $l/4$ Stellen hat. Mehr zu diesem Thema findet man im Übersichtsartikel [Gra08].

Wählen wir also ein (noch zu bestimmendes) festes u und für B alle Primzahlen $\leq p^{1/u}$, dann benötigen wir durchschnittlichen u^u Versuche bis eine Relation gefunden wird. Die Laufzeit ist dann gegeben durch die Anzahl der durchschnittlichen Versuche u^u um eine Relation zu finden mal der Anzahl der Divisionen pro Versuch $|B|$ mal der Anzahl der Relationen $|B|$ die man benötigt bevor man versuchen kann zu lösen. Insgesamt also $u^u|B|^2$. Nach dem Primzahlsatz (Satz 5.47) gilt $|B| \approx p^{1/u}/\log(p^{1/u})$ und somit muss u so gewählt werden, dass die Laufzeit

$$\frac{u^{u+2}p^{2/u}}{\log(p)}$$

minimiert wird. Die Ableitung ist

$$\frac{u^{u+2}p^{2/u}}{u^2\log(p)}\left(u^2\log(u) + u^2 + 2u - 2\log(p)\right)$$

und wir sehen, dass es eine eindeutige Nullstelle gibt, die mit p wächst. Es ist zwar nicht möglich diese Nullstelle exakt zu bestimmen, wenn wir aber

$$u \approx 2\sqrt{\frac{\log(p)}{\log(\log(p))}}$$

wählen, dann ist die Gleichung bis auf Terme niedrigerer Ordnung erfüllt. Die optimale Wahl für die Größe der Faktorbasis ist also $|B| = p^{1/u} \approx \exp(\log(p)/u) = \exp(\frac{1}{2}\sqrt{\log(p)\log(\log(p))})$ und $u^u = \exp(u\log(u)) \leq \exp(\sqrt{\log(p)\log(\log(p))})$. Somit ist die erwartete Laufzeit insgesamt

$$O(\exp(2\sqrt{\log(p)\log(\log(p))})$$

und das ist zwar nicht polynomial, aber besser als exponentiell.

Im Gegensatz zu den vorherigen Methoden funktioniert der Index Calculus nur in $\mathbb{Z}_p^*$ und ist daher auf anderen Gruppen, wie z.B. auf elliptischen Kurven (zu denen wir noch kommen) nicht effektiv anwendbar.

Das Problem ist, dass wir in einer allgemeinen Gruppe keine Primzahlen haben und damit weder klar ist was eine gute *Faktorbasis* wäre, bzw. wie die Zerlegung bezüglich der Faktorbasis (effektiv) bewerkstelligt werden soll.

Inzwischen gibt es auch verbesserte Varianten (Funktionenkörpersieb, Zahlenkörpersieb), die aber alle auf $\mathbb{Z}_p^*$ beschränkt sind. Der Rekord aus 2016 liegt bei p mit Bitlänge 768; p sollte daher deutlich länger sein (siehe Spalte 3 in Tabelle 2.2) und $p-1$ muss einen großen Primfaktor enthalten, damit die Pohlig-Hellman-Reduktion keine Vorteile bringt.

5.3.5 Shor Algorithmus

Zuletzt erwähnen wir noch, dass das DLP auf einem Quantencomputer effektiv gelöst werden kann. Es sei $G = \langle a \rangle$ und $\operatorname{ord}(a) = m$. Es soll $\log_a(b)$ für $b \in G$ gefunden werden. Dazu betrachtet man die Funktion

$$f : \mathbb{Z}_m \times \mathbb{Z}_m \to G, \qquad j = (j_1, j_2) \mapsto a^{j_2} \circ b^{-j_1}.$$

Die Funktion ist periodisch mit Periode $k = (1, \log_a(b))$, denn es gilt

$$f(j + k) = f(j), \qquad j \in \mathbb{Z}_m \times \mathbb{Z}_m.$$

Aber auch jedes Vielfache $k_r = (r, r \log_a(b))$, $0 \leq r < m - 1$ ist eine Periode und wir erhalten somit m verschiedene Perioden.

Perioden können mithilfe der Fouriertransformation gefunden werden und letztere kann effektiv von einem Quantencomputer berechnet werden. Wurde eine Periode k gefunden, so muss wegen $f(k) = n$ auch $k_2 + k_1 \log_a(b) = 0 \bmod m$ gelten. Ist k_1 invertierbar, so ist $\log_a(b) = k_2 k_1^{-1}$ gefunden. Das ist die Grundidee des **Shor Algorithmus** [Sho97a] (Peter Shor, *1959, amerikanischer Mathematiker) zur Lösung des DLP auf einem Quantencomputer. Wir werden darauf in Studienbrief 8 zurückkommen.

5.4 Der RSA-Verschlüsselungsalgorithmus

Zwei Jahre nach der Arbeit von Diffie und Hellman, also 1978, gelang es Ronald L. Rivest, Adi Shamir und Leonard Adleman eine Einweg-Falltürfunktion zu finden. Damit ergab sich der wohl bekannteste Public-Key-Algorithmus, der nach ihnen benannte **RSA-Algorithmus**:

- **Schlüsselerzeugung:** Alice wählt zwei verschiedene geheime Primzahlen p, q und bildet daraus die Zahlen $n = pq$ und $m = \text{kgV}(p-1, q-1)$. Als nächstes wählt sie eine Zahl e, die teilerfremd zu m ist, und berechnet die Zahl d, die $ed = 1 \pmod{m}$ erfüllt (das multiplikative Inverse von e modulo m). Die Zahlen (n, e) gibt Alice als öffentlichen Schlüssel bekannt. Die Zahl d behält sie als geheimen (= privaten) Schlüssel. Die Zahlen p, q und m bleiben geheim, werden aber nicht mehr benötigt.

- **Verschlüsselung:** Möchte nun Bob eine verschlüsselte Nachricht an Alice schicken, so schlägt er ihren öffentlichen Schlüssel (n, e) nach, verschlüsselt den Klartext x gemäß
$$y = x^e \pmod{n}$$
und schickt den Geheimtext y an Alice.

- **Entschlüsselung:** Zum Entschlüsseln verwendet Alice ihren geheimen Schlüssel (n, d) und berechnet damit den Klartext gemäß
$$x = y^d \pmod{n}.$$

Dass wirklich $y^d = (x^e)^d = x^{ed} = x \pmod{n}$ gilt, sieht man wie folgt: Da p und q teilerfremd sind, gilt $y^d = x \pmod{n}$ genau dann wenn $y^d = x \pmod{p}$ und $y^d = x \pmod{q}$ gilt. Da m ein Vielfaches von $p-1$ ist, folgt $ed = 1 \pmod{(p-1)}$ und es folgt $x^{ed} = x \pmod{p}$ aus dem kleinen Satz von Fermat (Satz 1.35) falls x kein Vielfaches von p ist. Ist x ein Vielfaches von p, so gilt $x^{ed} = x \pmod{p}$ ebenfalls, da nun beide Seiten gleich Null sind. Analog folgt $y^d = x \pmod{q}$ und damit die Behauptung.

RSA-Verschlüsselung

Alice		Bob
wählt p, q $n = p \cdot q$, $m = \text{kgV}(p-1, q-1)$ wählt e teilerfremd zu m $d = e^{-1} \pmod{m}$		
	n, e = öff. Schl. Alice	
		Nachricht x $y = x^e \pmod{n}$
	$\longleftarrow$ y	
$x = y^d \pmod{n}$		

Dass korrekt entschlüsselt wird folgt (da $ed = 1 + km$ für irgendein $k \in \mathbb{N}_0$) aus

$$y^d = (x^e)^d = x^{ed} = x^{1+km} = x \pmod{n}.$$

In der Tat, ist x teilerfremd zu n, so folgt der letzten Schritt wegen $\lambda(n) = m$ aus Satz 5.28. Ist x nicht teilerfremd zu n, so ist $x = 0 \pmod{n}$ und die Behauptung folgt unmittelbar.

Oft wird bei der Formulierung des Algorithmus statt $m = \lambda(n) = \mathrm{kgV}(p-1, q-1)$ alternativ $m = \varphi(n) = (p-1)(q-1)$ verwendet. Damit klappt der Algorithmus natürlich auch. Da aber $\lambda(n) < \varphi(n)$ ist, funktionieren in diesem Fall die im Allgemeinen kleineren Parameter $e' = e \bmod \lambda(n)$ und $d' = d \bmod \lambda(n)$ ebenfalls, denn es gilt ja $y = x^e = x^{e'} \pmod{n}$ und $x = y^d = y^{d'} \pmod{n}$.

Warum kann die neugierige Eve hier nichts ausrichten? Eve kennt den öffentlichen Schlüssel n, e und kann $y = x^e \pmod{n}$ belauschen. Das Potenzieren mit e modulo n ist eine Einweg-Falltürfunktion. Sie kann für hinreichend großes n praktisch nicht umgekehrt werden (Einwegeigenschaft). Eine Geheiminformation, mit der sie leicht umgekehrt werden kann, ist der geheime Schlüssels d (Falltüreigenschaft). Eve kann nun natürlich prinzipiell den geheimen Schlüssel d berechnen, indem sie die Gleichung

$$de = 1 \pmod{m}$$

nach d löst. Um das zu tun, muss sie aber erst $m = \mathrm{kgV}(p-1, q-1)$ kennen, also die Primfaktoren p und q von n finden. Sind diese beiden Primfaktoren jedoch geeignet gewählt (insbesondere genügend groß), so wird aber auch der heutzutage schnellste Computer das Zeitliche segnen, bevor er mit der Primfaktorzerlegung von n fertig ist.

Nun gleich zu einem Beispiel (die Primzahlen sind klein gewählt, damit wir im Kopf rechnen können):

Beispiel 5.37 RSA-Algorithmus
a) Wie erzeugt Alice aus dem Primzahlpaar $p = 5$, $q = 11$ einen öffentlichen und einen privaten Schlüssel?
b) Bob möchte Alice die Nachricht $x = 8$ verschlüsselt schicken. Wie muss er vorgehen?
c) Wie enschlüsselt Alice die Nachricht von Bob?

Lösung zu 5.37

a) Aus $p = 5$ und $q = 11$ berechnet Alice

$$n = pq = 55$$

das ist bereits ein Teil des öffentlichen Schlüssels. Für den zweiten Teil berechnet sie

$$m = \mathrm{kgV}(p-1, q-1) = 20$$

und weiters muss sie eine Zahl e wählen, die teilerfremd zu m ist. Sie wählt z.B. $e = 3$. Der geheime Schlüssel ist nun das multiplikative Inverse von e modulo m, erfüllt also

$$3d = 1 (\mathrm{mod}\, 20)$$

Der erweiterte Euklid'sche Algorithmus (oder der Computer;-) liefert $d = 7$. Nun gibt Alice

$$(n, e) = (55, 3)$$

als öffentlichen Schlüssel bekannt und behält

$$(n, d) = (55, 7)$$

als privaten Schlüssel (zwar ist nur d geheim, man fasst aber oft auch n als Bestandteil des privaten Schlüssels auf, da es ja zum Entschlüsseln notwendig ist.)

b) Bob verschlüsselt $x = 8$, indem er die Zahl mit Alices öffentlichem Schlüssel $e = 3$ potenziert und den Rest modulo n berechnet:

$$y = x^e = 8^3 = 17 \pmod{55}.$$

c) Alice entschlüsselt $y = 17$, indem sie mit ihrem geheimen Schlüssel $d = 7$ potenziert und den Rest modulo n nimmt:

$$x = y^d = 17^7 = 8 \pmod{55}.$$

■

Martin Gardner hat das RSA-Verfahren erstmals 1977 in seiner Kolumne „Mathematische Spiele“ für den *Scientific American* vorgestellt. Er hat einen verschlüsselten Text gemeinsam mit dem öffentlichen Schlüssel abgedruckt. Es hat siebzehn Jahre gedauert, bis die Primfaktoren gefunden waren (von 600 Freiwilligen) und Gardners Nachricht entschlüsselt werden konnte: THE MAGIC WORDS ARE SQUEAMISH OSSIFRAGE (*Die magischen Worte sind zimperliche Lämmergeier*). Gardner hatte übrigens $n = pq$ in der Größenordnung 10^{129} verwendet.

Zum Verschlüsseln längerer Nachrichten, muss die Nachricht in Blöcke der Größe $< n$ zerlegt werden und es kann dann ECB oder CBC verwendet werden (OFB, CFB, CTR scheiden aus, da bei diesen Modi mit der Verschlüsselungsfunktion entschlüsselt werden kann).

Dabei ist es natürlich wichtig die maximale Blockgröße auszunutzen! Wenn Sie buchstabenweise im ASCII-Code verschlüssen, dann kann diese monoalphabetische Verschlüsselung (unanbhängig vom verwendeten Algorithmus) leicht mit statistischen Methoden gebrochen werden.

Beispiel 5.38 RSA-Algorithmus
Alice hat den öffentlichen Schlüssel $(n, e) = (1147, 29)$.
a) Bob möchte ihr die Nachricht 251782167 verschlüsselt schicken. Wie geht er vor?
b) Der geheime Schlüssel von Alice ist $d = 149$. Wie entschlüsselt sie die Nachricht?

Lösung zu 5.38

a) Bob zerlegt die Nachricht in Blöcke gleicher Länge, so, dass die entstehenden Zahlen kleiner n sind: $0251, 0782, 0167$. Dann verschlüsselt er jede Zahl für sich:

$$\begin{aligned} y_1 &= 251^{29} = 1013 \pmod{1147} \\ y_2 &= 782^{29} = 257 \pmod{1147} \\ y_3 &= 167^{29} = 757 \pmod{1147} \end{aligned}$$

Somit lautet die geheime Nachricht: 101302570757.

Wir haben hier die Blöcke dem Dezimalsystem angepasst gewählt. Bei der Implementation im Computer wird man natürlich das Dualsystem verwenden. Möchte man die maximale Blockgröße auszunutzen, muss man die Nachricht im Zahlensystem zur Basis n zerlegen: $251782167 = (191, 436, 756)_{1147}$.

b) Alice zerteilt die Nachricht in Blöcke der Länge 4 und entschlüsselt jeden Block mithilfe ihres geheimen Schlüssels $d = 149$:

$$\begin{aligned} x_1 &= 1013^{149} = 251 \pmod{1147} \\ x_2 &= 257^{149} = 782 \pmod{1147} \\ x_3 &= 757^{149} = 167 \pmod{1147} \end{aligned}$$

■

Die Berechnung der Potenz x^e modulo n erfolgt nach dem **Square-and-Multiply**-Verfahren und das geht am schnellsten, wenn e nur wenige Stellen im Binärsystem hat. In der Praxis ist daher oft als öffentlicher Schlüssel eine **Fermat-Zahl** $F_k = 2^{2^k} + 1$ in Verwendung.

Beispiel 5.39 Die ersten Fermat-Zahlen sind

$$F_0 = 3, \quad F_1 = 5, \quad F_2 = 17, \quad F_3 = 257, \quad F_4 = 65\,537, \quad F_5 = 4\,294\,967\,297.$$

Fermat vermutete, dass F_k immer prim ist, aber Euler gelang es $F_5 = 6\,700\,417 \cdot 641$ zu faktorisieren.

Bei zu kleinen Werten (z.B. $e = 3$) ist eine **Low-Exponent-Attacke** (siehe nächster Abschnitt) möglich, aber (z.B.) $e = F_4 = 65\,537$ ist eine gängige Wahl (in [FIPS186-5] wird $2^{16} < e < 2^{256}$ gefordert). Das ist kein Sicherheitsproblem, es dürfen nur nicht dieselben Primfaktoren p und q verwendet werden.

Aufgrund seiner einfachen algebraischen Struktur ist RSA anfällig für eine Reihe von Angriffen, von denn wir einige im nächsten Abschnitt besprechen werden. Deshalb ist es für einen sicheren Einsatz notwendig, ein kryptographisch sicheres Padding der Nachrichtneblöcke zu verwenden (siehe [RFC8017] bzw. nächster Abschnitt).

Das Entschlüsseln kann man beschleunigen indem man $x \bmod p = y^d \bmod p$, $x \bmod q = y^d \bmod q$ berechnet und dann x mithilfe des Chinesischen Restsatz bestimmt.

Beispiel 5.40 Beschleunigte Entschlüsselung
Entschlüsseln sie den ersten Block $y = 1013$ aus dem letzten Beispiel mit mithilfe des Chinesischen Restsatz.

Lösung zu 5.40 Es gilt $n = p_1 p_2$ mit $p_1 = 31$ und $p_2 = 37$. Wir zerlegen den geheimen Schlüssel in die zwei Teile $d_1 = d \bmod (p_1 - 1) = 29$, $d_2 = d \bmod (p_2 - 1) = 5$ und bestimmen die multiplikativen Inversen $q_1 = p_1^{-1} \pmod{p_2} = 6$, $q_2 = p_2^{-1} \pmod{p_1} = 26$, die wir für den Chinesischen Restsatz benötigen. Nun berechnen wir

$$x_1 = y^{d_1} = 3 \pmod{p_1}, \qquad x_2 = y^{d_2} = 29 \pmod{p_2}$$

und $x = x_1 p_2 q_2 + x_2 p_1 q_1 = 251 \pmod{n}$.

Setzt man $p_1 < p_2$ voraus, so gilt sogar $x = x_1 + (x_2 - x_1) p_1 q_1 \pmod{n}$, denn man rechnet leicht $x = x_j \pmod{p_j}$ nach. Das spart eine Multiplikation und man braucht q_2 nicht.

■

Sowohl bei Software- als auch bei Hardware-Implementierungen ist RSA wesentlich langsamer als symmetrische Verfahren wie z.B. AES. In der Praxis ist der RSA-Algorithmus daher meist zu aufwendig und wird *nur zum Austausch des geheimen Schlüssels* eines konventionellen symmetrischen Verschlüsselungsalgorithmus (AES, ...) verwendet. Solche „gemischten" Verfahren (zunächst RSA zum sicheren Austausch eines geheimen Sitzungsschlüssels, danach Verschlüsselung der Nachricht mit einem symmetrischen Verfahren) werden **Hybridverfahren** genannt.

Wie man passende RSA-Schlüssel erzeugt werden wir in Abschnit 5.6 besprechen. Wir erwähnen aber schon jetzt, dass die Erzeugung einen gewissen Aufwand bedeutet. Deshalb ist DH für Sitzungsschlüssel (die nur einmal verwendet werden, vgl. PFS) besser geeignet.

Ausserdem ist es möglich, für n das Produkt von mehr als zwei Primzahlen zu verwenden (solange alle Primzahlen verschieden sind) und das ist als **Multi-Prime RSA** bekannt. Solange man $m = \lambda(n)$ (oder $m = \varphi(n)$) verwendet, bleibt alles andere unverändert (Aufgabe 28). Der Hauptvorteil liegt darin, dass man die Entschlüsselung, wie in Beispiel 5.40 beschrieben, noch weiter beschleunigen kann. Im Standard [RFC8017] ist auch diese Möglichkeit vorgesehen.

5.5 Sicherheit des RSA-Algorithmus

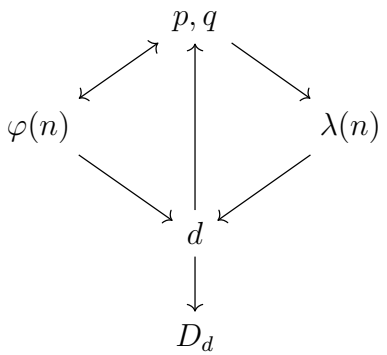
Wir haben gesehen, dass der RSA-Algorithmus geknackt ist, sobald die Faktorisierung von $n = pq$ gelingt. Damit die Faktorisierung nicht (innerhalb praktischer Zeitgrenzen) gelingt, müssen die Primfaktoren p und q von n daher hinreichend

groß sein. Die Länge des Moduls n wird dabei als **Schlüssellänge** bezeichnet. In der Praxis werden Schlüssellängen von zumindest 1024 verwendet (das entspricht dem Produkt zweier 512 Bit Primzahlen). Da ein zu großer Schlüssel nur sinnlos den Rechenaufwand erhöht, wird die Schlüssellänge abhängig vom Zeitraum, über den hinweg die Daten gesichert werden müssen und abhängig vom Wert der Daten gewählt. Eine Schlüssellänge, die heute als sicher gilt, kann in einigen Jahren vielleicht schon (mit entsprechenden Ressourcen) geknackt werden. Das bedeutet, dass Daten, die heute mit dieser Schlüssellänge verschlüsselt (oder signiert) werden, nicht auf ewig sicher verschlüsselt bzw. signiert sind!

Seit dem Jahr 1991 gab es einen internationalen Wettbewerb der Firma RSA-Security zur Faktorisierung großer RSA-Zahlen (= Produkte zweier etwa gleich großer Primzahlen). Für die Faktorisierung vorgegebenen Zahlen verschiedener Länge (von 576 bis 2048 Bit) waren mit der Länge ansteigende Preisgelder bis zu US$ 200 000,- ausgesetzt. Der Wettbewerb wurde 2007 abgebrochen. Die höchste bis dahin faktorisierte Zahl war RSA-640. Obwohl es kein Preisgeld mehr gibt, gehen die Bemühungen die fehlenden Zahlen zu faktorisieren weiter. RSA-250 (829 Bits) wurde 2020 faktorisiert und es ist wohl absehbar bis man 1024 Bit RSA Zahlen faktorisieren kann. Dementsprechend wird empfohlen Schlüssel dieser Länge nicht mehr lange zu verwenden.

Die **Zerlegung von n in Primfaktoren** (und die dann mögliche Berechnung des privaten Schlüssels d) ist der *einzige heute bekannte* universelle Angriff auf RSA. Die Primfaktorzerlegung ist nach heutigem Wissensstand ein sehr schwieriges Problem. Es hängt (sub-)exponentiell von der Anzahl der Stellen von n ab. Es könnte jedoch sein, dass es einen effizienten, d.h., polynomialen Algorithmus zur Faktorisierung gibt, und in diesem Fall wäre die Sicherheit des RSA-Algorithmus endgültig dahin. Ein wichtiges offenes Problem ist es deshalb zu *beweisen*, dass es einen solchen Algorithmus nicht geben kann (das ist jedoch trotz jahrhundertelanger Forschung noch nicht gelungen).

Folgendes Diagramm veranschaulicht, welche Größen bei bekannten (n, e) (effizient) auseinander berechnet werden können:



Dabei ist d als irgendein Schlüssel mit dem die Entschlüsselung funktioniert zu verstehen. Das Diagramm impliziert, dass die Bestimmung einer der Größen $\varphi(n)$, $\lambda(n)$ oder d aus dem öffentlichem Schlüssel (n, e) algorithmisch nicht einfacher möglich sein kann als die Faktorisierung von n.

Wie man aus der Faktorisierung $\varphi(n)$ bzw. $\lambda(n)$ und daraus dann d berechnet wissen wir schon. Die restlichen Pfeile realisiert man wie folgt:

- $\varphi(n) \to p, q$: Kennt man $\varphi(n)$, so kann man n faktorisieren:

$$p, q = \frac{1}{2}\big(n - \varphi(n) + 1 \pm \sqrt{(n - \varphi(n) + 1)^2 - 4n}\big).$$

- $d \to p, q$: Kennt man e und d, so kann man n faktorisieren:

  Es gilt ja $ed - 1 = t\lambda(n)$ und schreiben wir $ed - 1 = 2^j k$ mit k ungerade, so gilt $a^{2^j k} = 1 \pmod{n}$ für alle $a \in \mathbb{Z}_n^*$. Die Ordnung von a^k muss also 2^l mit $0 \leq l \leq j$ sein. Nun kann man, wie in Abschnitt 5.5.6 beschrieben wird, n faktorisieren.

Diese Überlegungen zeigen aber nicht, dass es nicht doch einen einfachen Weg gibt, den Klartext aus dem Geheimtext zu bestimmen (ohne d zu bestimmen). Das wäre der fehlende Pfeil von D_d nach d. Eine solche Möglichkeit wäre es, direkt aus dem Geheimtext $y(= x^e)$ die e-te Wurzel zu ziehen, um an x zu kommen. Das ist äquivalent dazu, das Polynom $p(x) = x^e - y$ über $\mathbb{Z}_n$ zu faktorisieren. Für diese Faktorisierung sind aber nur im Falle eines endlichen Körpers effektive Algorithmen bekannt. Um das Problem mithilfe des Chinesischen Restsatzes auf den Fall eines Körpers zurückzuführen, brauchen wir aber die Faktorisierung von n.

Nach aktuellem Wissenstand, ist das Entschlüsseln also nur bei bekannter Faktorisierung von n möglich. Trotzdem gibt es eine Anzahl von Angriffsmöglichkeiten über die man sich beim Einsatz von RSA bewußt sein sollte:

(1) **Common-Modulus-Angriff**: Alice und Bob dürfen nicht das gleiche n verwenden, da beide die Primfaktoren bestimmen können und damit den geheimen Schlüssel des anderen. Das geht sogar ohne n zu faktorisieren. Sind (n, e'), (n, d') die Schlüssel von Alice und (n, e), (n, d) die Schlüssel von Bob, so gilt

$$d = e^{-1} \pmod{\frac{e'd' - 1}{\mathrm{ggT}(e, e'd' - 1)}}.$$

Wegen $\frac{e'd'-1}{\mathrm{ggT}(e,e'd'-1)} = t\lambda(n)$ erhalten wir zwar den Schlüssel von Bob nur bis auf ein Vielfaches von $\lambda(n)$, entschlüsselt wird damit aber in jedem Fall korrekt.

Wird die gleiche Nachricht an zwei Teilnehmer mit gleichem n geschickt, so kann die Nachricht aus den beiden Geheimtexten berechnet werden (Aufgabe 26).

(2) **Low-Exponent-Angriff**: Ist $x^e < n$ in $\mathbb{N}$, so kann $x = \sqrt[e]{y}$ durch Wurzelziehen in $\mathbb{N}$ berechnet werden. Das funktioniert auch ohne die Einschränkung wenn mehrere Personen den gleichen (kleinen) öffentlichen Schlüssel e verwenden. Angenommen Alice schickt ihre Geburstagseinladung an ihre Freunde, die alle den gleichen Exponenten e verwenden. Der gleiche Klartext x wurde also mit verschiedenen öffentlichen Schlüsseln (n_1, e), ..., (n_k, e) verschlüsselt, wobei die n_1, ...,

n_k paarweise teilerfremd sind. Es gilt also

$$y_j = x^e \pmod{n_j}.$$

Ist nun $x^e \leq n_1 \cdots n_k$, so können wir $y = x^e$ mit dem Chinesischen Restsatz ermitteln und daraus $x = \sqrt[e]{y}$ berechnen. Bei $e = 3$ würde es also reichen, wenn die gleiche Nachricht an 3 verschiedene Empfänger verschickt wird. Für $e = 3$ gibt es übrigens noch weitere Angriffe (siehe [Hin09]). Eine Erweiterung ist der **Håstad-Broadcast-Angriff**, wenn ähnliche Nachrichten an mehrere Empfänger geschickt werden. Diese Angriffe werden durch kryptographisches Padding effektiv verhindert.

(3) **Stereotyped-Message-Angriff**: Ebenfalls problematisch ist es, wenn ein großer Teil der Nachricht bekannt ist, da der fehlende Teil dann mit Brute-Force ermittelt werden kann (indem man alle Möglichkeiten für den fehlenden Teil mit E_e verschlüsselt und testet ob sich der gewünschte Geheimtext ergibt). Wenn z.B. eine Bank einen standardisierten Text mit der vierstelligen PIN verschlüsselt an ihre Kunden verschickt. Kryptographisches Padding verhindert auch diesen Angriff.

Es geht aber auch ohne Brute-Force. Kennt man alle Bits der Nachricht bis auf einen Bruchteil von maximal $1/e$ aufeinanderfolgenden Bits, so kann der Rest berechnet werden. Dazu schreiben wir die Nachricht als $x = b + 2^l u$, wobei b die bekannten Bits und $u < 2^e$ die unbekannten Bits sind. Dann ist u eine *kleine* Nullstelle des Polynoms

$$(b + 2^l X)^e - y \in \mathbb{Z}_n[X].$$

Wie bereits erwähnt, ist es im Allgemeinen nicht möglich die Nullstellen ohne Kenntnis der Faktorisierung von n effektiv zu bestimmen. Allerdings gibt es den **Coppersmith-Algorithmus**, mit dem für ein Polynom vom Grad r alle Nullstellen kleiner $\sqrt[r]{n}$ effektiv gefunden werden können. Wir werden das in Abschnitt 9.2.2 besprechen.

Ein ähnliches Problem ergibt sich, wenn zwischen zwei Nachrichten eine affine Beziehung besteht: $x_2 = ax_1 + b$ (**Franklin–Reiter-Angriff**). Dann ist x_1 eine gemeinsame Nullstelle der beiden Polynome

$$X^e - y_1, \quad (aX + b)^e - y_2 \in \mathbb{Z}_n[X]$$

und kann durch Bilden des größten gemeinsamen Teilers gefunden werden. Für weitere Details siehe wieder [Hin09].

(4) **Cycling-Angriff**: Die Menge aller Verschlüsselungsfunktionen $G = \{E_e | e \in \mathbb{Z}^*_{\lambda(n)}\}$ bei RSA (also $E_e(x) = x^e \pmod{n}$) ist selbst wieder eine kommutative Gruppe, wenn man die Hintereinanderausführung als Gruppenverknüpfung nimmt. Das neutrale Element ist E_1, das inverse Element ist die Umkehrfunktion $(E_e)^{-1} = D_d$ und aufgrund der Potenzgesetze gilt ja $E_e \circ E_{e'} = E_{ee'}$. Insbesondere die letzte Formel besagt, dass die Abbildung $\psi : \mathbb{Z}^*_{\lambda(n)} \to G$, $e \mapsto E_e$ ein Gruppenisomorphismus ist (Surjektivität ist klar und Injektivität folgt aus Satz 5.28 — hier ist

es wichtig, dass wir mit $\lambda(n)$ und nicht mit $\varphi(n)$ arbeiten, denn für $e, e' \in \mathbb{Z}^*_{\varphi(n)}$ gilt $E_e = E_{e'}$ genau dann wenn $e = e' \pmod{\lambda(n)}$. Also gilt $|G| = \varphi(\lambda(n))$ und nach Satz 5.28

$$e^{\lambda(\lambda(n))} = 1 \qquad \text{bzw.} \qquad d = e^{\lambda(\lambda(n))-1}.$$

Die kleinste Potenz mit der das klappt ist gerade die Ordnung von e in $\mathbb{Z}^*_{\lambda(n)}$. Im Prinzip erhält man den Klartext also immer indem man den Geheimtext oft genug mit dem öffentlichen Schlüssel verschlüsselt. Die Ordnung ist die Potenz bei der man für alle Nachrichten x wieder zu x zurückkehrt. Für ein einzelnes x kann das aber auch schon früher passieren und die kleinste Potenz wird als **Wiederherstellungsexponent** bezeichnet.

Wenn man wissen möchten, wie viele $x \in \mathbb{Z}_n$ die Gleichung $x^k = x$ erfüllen, dann reicht es aufgrund der Isomorphie $\mathbb{Z}_n \cong \mathbb{Z}_p \times \mathbb{Z}_q$ diese Frage für $\mathbb{Z}_p$ zu beantworten. In diesem Fall gilt $\{x \in \mathbb{Z}_p | x^k = x\} = 1 + \{x \in \mathbb{Z}_p^* | x^{k-1} = 1\} = 1 + \text{ggT}(p-1, k-1)$ (Bemerkung nach Satz 5.20). Also $\{x \in \mathbb{Z}_m | x^k = x\} = (1 + \text{ggT}(p-1, k-1))(1 + \text{ggT}(q-1, k-1))$. Also gibt es genau $(1+\text{ggT}(e^r-1, p-1))(1+\text{ggT}(e^r-1, q-1))$ Werte x deren Wiederherstellungsexponent r teilt. Da e ungerade ist, gibt es also immer mindestens 9 Elemente mit Wiederherstellungsexponent 1.

Der Cycling-Angriff hat aber keine praktische Bedeutung, da man zeigen kann, dass für fast alle Parameter (p, q, e) die meisten x einen großen Wiederherstellungsexponenten haben (siehe [FPS01]).

(5) **Wiener-Angriff**: Ein zu kleiner Wert für d muss vermieden werden. Gilt $d < n^{1/4}/3$, so ist folgender Angriff möglich: Wir betrachten den einfacheren Fall bei dem e und d mittels $\varphi(n)$ (statt $\lambda(n)$) bestimmt wurden. Dann gilt

$$ed = 1 + k\varphi(n)$$

für ein $k \in \mathbb{N}$. Nun formen wir diese Gleichung so um, dass auf der linken Seite möglichst nur noch öffentlich bekannte Informationen stehen:

$$\frac{e}{\varphi(n)} = \frac{k}{d} + \frac{1}{d\varphi(n)}.$$

Da Eve $\varphi(n)$ nicht kennt, versucht sie $\varphi(n)$ durch n zu ersetzten:

$$\frac{e}{n} = \frac{k}{d} + \frac{e}{n\varphi(n)}\left(\frac{1}{d} - n + \varphi(n)\right) = \frac{k}{d} + O(n^{-1/2}).$$

Diese Gleichung können wir nun wie folgt interpretieren: Ist d klein, so kann der Bruch e/n gut durch den Bruch k/d mit kleinerem Nenner approximiert werden. Solche Approximationen mit kleinen Nenner findet man aber mittels **Kettenbruchentwicklung**.

Die Darstellung

$$\frac{m}{n} = b_0 + \cfrac{1}{b_1 + \cfrac{1}{b_2 + \cfrac{1}{\ddots \; b_{k-1} + \frac{1}{b_k}}}}$$

bezeichnet man als Kettenbruchentwicklung der Zahl m/n. Sie kann berechnet werden indem man mit $x_0 = m/n$ startet und dann rekursiv $b_j = \lfloor x_j \rfloor$, $x_{j+1} = (x_j - b_j)^{-1}$ bildet. Die Koeffizienten b_j sind übrigens genau die Quotienten die beim Euklid'schen Algorithmus bei der Berechnung von $\mathrm{ggT}(n, m)$ auftreten. Insbesondere hat jede rationale Zahl eine endliche Kettenbruchentwicklung.

Bricht man die Entwicklung nach j Koeffizienten ab, so erhält man eine Approximation $\frac{m}{n} \approx \frac{m_j}{n_j}$ die als $j$ter Näherungsbruch (oder $j$te **Konvergente**) bezeichnet wird. Der $j$te Näherungsbruch kann rekursiv durch $m_j = b_j m_{j-1} + m_{j-2}$, $n_j = b_j n_{j-1} + n_{j-2}$ mit Startwerten $m_{-1} = n_{-2} = 1$, $m_{-2} = n_{-1} = 0$ berechnet werden und es gilt

$$\frac{m_{j-1}}{n_{j-1}} - \frac{m_j}{n_j} = \frac{(-1)^j}{n_j n_{j-1}}.$$

Das sieht man, indem man die Rekursion mit Induktion beweist und damit $n_j m_{j-1} - m_j n_{j-1} = (-1)^j$ zeigt. Insbesondere folgt daraus

$$\left| \frac{m}{n} - \frac{m_j}{n_j} \right| \leq \frac{1}{n_j n_{j+1}} \leq \frac{1}{n_j^2},$$

da der nächste Wert immer zwischen den beiden vorherigen Werten liegt. Man kann zeigen, dass die $j$ten Näherungsbrüche die beste Approximation unter allen Brüchen mit Nenner $\leq n_j$ ist und dass, wenn

$$\left| \frac{m}{n} - \frac{\tilde{m}}{\tilde{n}} \right| \leq \frac{1}{2\tilde{n}^2}$$

gilt, dann muss $\tilde{m}/\tilde{n}$ ein $j$ter Näherungsbruch sein (siehe [Har+08] Theorem 181, 184).

Die Idee ist also, den geheimen Schlüssel in der Kettenbruchentwicklung von e/n zu suchen:

Beispiel 5.41 Führen Sie den Wiener-Angriff auf den öffentlichen Schlüssel $(n, e) = (2021027, 1614413)$ durch.

Lösung zu 5.41 Die approximierenden Kettenbrüche sind

$$1, \frac{3}{4}, \frac{4}{5}, \frac{131}{164}, \ldots, \frac{1614413}{2021027}$$

und wir testen ob der Schlüssel unter den Nennern ist: Alle geraden Kandidaten scheiden aus. Für die ungeraden prüfen wir ob $ed = 1 \pmod{k}$ gilt. Wenn ja, dann haben wir einen potentiellen Kandidaten.

Somit scheidet 4 aus. Wegen $1614413 * 5 - 1 = 0 \pmod 4$ ist 5 im Rennen. Ist $d = 5$ richtig, so wäre $\varphi(n) = \frac{ed-1}{k} = 2018016$. Damit können wir versuchen n wie oben beschrieben zu faktorisieren: $p, q = \frac{1}{2}(n - \varphi + 1 \pm \sqrt{(n - \varphi + 1)^2 - 4n}) = 2003, 1009$ und somit sind wir fertig. ■

Damit d in der Liste der Kettenbrüche auftaucht, muss der Fehler kleiner als $1/(2d^2)$ sein, also $d \leq Cn^{1/4}$ (indem man den Fehlerterm genauer abschätzt, sieht man, dass man $C = 1/3$ wählen kann).

Wurden e und d mittels $\lambda(n)$ bestimmt, so gilt $\varphi(n) = t\lambda(n)$ und unsere Gleichung ändert sich zu

$$\frac{e}{n} = \frac{k}{td} + \frac{e}{n\varphi(n)} \left(\frac{1}{d} - n + \varphi(n) \right) = \frac{k}{td} + O(n^{-1/2}).$$

Nun sind aber k und t nicht mehr unbedingt teilerfremd und in der Liste der Kettenbrüche taucht (wenn überhaupt) nur die gekürzte Form auf. Das erschwert den Angriff erheblich. Wenn wir hoffen, dass sich nichts kürzt, dann könnte man aus $e(td) = t + k\varphi(n)$ sowohl t als auch $\varphi(n)$ bestimmen und damit versuchen n zu faktorisieren. Wir wollen das hier aber nicht weiter verfolgen und erwähnen nur, dass es auch noch eine Verbesserung, den **Boneh–Durfee-Angriff**, gibt.

Wählt man e klein (zur Beschleunigung der Verschlüsselung), so ist in jedem Fall d groß (wegen $k \geq 1$ gilt ja $d > \frac{\varphi(n)}{e}$ bzw. $d > \frac{\lambda(n)}{e}$). Man muss aber vorsichtig sein mit zu einschränkenden Bedingungen an d (z.B. eine Zahl mit wenigen 1 in der Binärdarstellung, damit schneller entschlüsselt werden kann), da man immer davon ausgehen muss, dass diese Kriterien bekannt werden.

(6) **Camouflage-Angriff**: Um die Nachricht $y = x^e$ zu entschlüsseln, kann Mallory $r \in \mathbb{Z}_n^*$ wählen und Alice bitten die unverdächtige Nachricht $y' = yr^e$ zu entschlüsseln. Mit dem Ergebnis $x' = xr$ kann Mallory dann $x = r^{-1}x'$ berechnen. Entschlüsseln Sie also niemals unbekannte Nachrichten und geben das Ergebnis bekannt!

(7) **Seitenkanalangriff**: Es gibt auch nicht-mathematische Angriffe: Die notwendigen Rechenschritte beim Entschlüsseln mittels Square-and-Multiply-Verfahren hängen von der Binärdarstellung des geheimen Schlüssel ab und dieser kann daher durch Messen der Zeit bzw. des Stromverbrauchs während der Entschlüsselung ermittelt werden.

Weitere Angriffe mit mathematischen Details findet man in [Bon99], [Hin09], [Kat01].

Um die beschriebenen Angriffe effektiv zu verhindern ist es für die Sicherheit entscheidend, die Verschlüsselung nur in Kombination mit einem kryptographisch sicherem Padding einzusetzen. In der Praxis wird das **Optimal Asymmetric Encryption Padding** (**OAEP**) verwendet, das im Wesentlichen ein Feistelnetzwerk mit zwei Hashfunktion als S-Box ist, das den zufälligen Paddingblock gleichmäßig auf den Klartext verteilt. Die Ver-/Entschlüsselung eines Blocks x läuft damit wie folgt ab:

- Es sei $2^{k+k_0} < n$ mit k der Blocklänge der Nachricht x (Nachricht und Padding dürfen die RSA Blockgröße nicht überschreiten). Gegeben sind zwei kryptographische Hashfunktionen $G : \mathbb{Z}_2^{k_0} \to \mathbb{Z}_2^k$ und $H : \mathbb{Z}_2^k \to \mathbb{Z}_2^{k_0}$.

- Zum Verschlüsseln wähle eine Zufallszahl (**Nonce**) $r \in \mathbb{Z}_2^{k_0}$ und berechne

$$X = x \oplus G(r), \qquad Y = r \oplus H(X).$$

  Der Geheimtext lautet:

$$y = E_e(X \| Y).$$

- Zum Entschlüsseln

$$X \| Y = D_d(y)$$

und

$$r = Y \oplus H(X), \quad x = X \oplus G(r).$$

Details dazu sind im Standard [RFC8017] beschrieben.

In der Praxis ist die Nachricht als eine Folge von Bytes (**Octet String**) gegeben und die Umwandlung zu einer Zahl in $\mathbb{Z}_n$ erfolgt indem man die Bytes in *umgekehrter* Reihenfolge als Ziffern im Zahlensystem zur Basis 2^8 betrachtet.

In der ursprünglichen Version des Standards wurde noch eine einfachere Version des Paddings empfohlen. Diese wurde vom Schweizer Kryptologen Daniel Bleichenbacher (*1964) gebrochen (**Bleichenbacher Angriff**), der damit das **Secure Sockets Layer** (SSL) Protokoll (der zentrale Grundstein verschlüsselter Kommunikation im Internet) erfolgreich angreifen konnte.

Als Alternative zu OAEP, und da RSA nur zur Schlüsselvereinbarung für einen symmetrischen Algorithmus verwendet wird, gibt es auch noch das **Schlüsselkapselungsverfahren** (*key encapsulation mechanism*, **KEM**). Abstrakt ist es wie folgt durch drei Funktionen definiert:

- *Generate*: Erzeugt ein Schlüsselpaar aus öffentlichem und privaten Schlüssel.
- *Encapsulate*: Erzeugt aus dem öffentlichen Schlüssel ein Geheimnis r und eine gekapselte (verschlüsselte) Nachricht c vom Geheimnis.
- *Decapsulate*: Ermittelt aus dem privaten Schlüssel und der gekapselten Nachricht c das Geheimnis r.

Die Formulierung als KEM dient der Standardisierung der Schnittstellen um möglichen Fehlern bei der Anwendung vorzubeugen und um Sicherheitsbeweise zu erleichtern.

Konkret wird bei RSA eine Zufallszahl $r \in \mathbb{Z}_n$ erzeugt und wie üblich verschlüsselt $c = r^e$. Das Entschlüsseln erfolgt natürlich ebenfalls wie üblich mit $r = c^d$. Der Vorteil ist, dass r maximale Größe hat und daher kein Padding notwendig ist. Da r in der Regel nicht die passende Größe für den verwendeten symmetrischen Algorithmus hat, gibt es noch eine **Schlüsselableitungsfunktion** (*key derivation function*, **KDF**), die aus r eine Zahl passender Bitlänge macht. In der Praxis basiert die KDF auf einer kryptographischen Hashfunktion.

Zum Abschluß beschreiben wir noch einige der klassichen Faktorisierungsalgorithmen, denn auch bei der Auswahl der Primfaktoren können Fehler passieren. Für Implementation dieser Verfahren am Computer und weitere mathematische Details sei auf [CP05; For15] verwiesen.

5.5.1 Sieb des Eratosthenes

Einer der einfachsten und ältesten Algorithmen zur Faktorisierung einer Zahl n ist das **Sieb des Eratosthenes**. Es beruht auf der Überlegung, dass einer der Primfaktoren $\leq \sqrt{n}$ sein muss und es daher genügt, alle Zahlen kleiner oder gleich

$\sqrt{n}$ zu prüfen. Man schreibt also alle Zahlen von 1 bis $\sqrt{n}$ an, und beginnt mit der kleinsten, 2, ob sie n teilt. Wenn nein, dann kann man 2 und alle Vielfachen von 2 aus der Liste streichen. Dann geht man zur nächstgrößeren ungestrichenen Zahl weiter usw. Der Rechenaufwand wächst hier exponentiell mit der Anzahl der Stellen von n.

Denn die Rechenzeit wächst proportional zu $\sqrt{n}$, und wenn n in Binärdarstellung b Bit lang ist, so ist $2^{b-1} \leq n < 2^b$, also insbesondere $n = O(2^b)$, also folgt für die Rechenzeit $T(b)$

$$T(b) \leq c \cdot \sqrt{n} = O(\sqrt{2}^b)$$

sie wächst also exponentiell mit der Zahl der Bits von n.

In der Praxis erstellt man sich eine Liste aller Primzahlen unter einer gegeben Schranke (z.B. 2^{32}) und verwendet als ersten Schritt die **Probedivision** mit diesen Primzahlen um kleine Primfaktoren zu finden.

- Weder p noch q sollte klein sein. Beide Primzahlen p und q sollten ungefähr gleich groß sein.

5.5.2 Fermat

Liegen die beiden Primzahlen zu dicht beieinander, so kann die Faktorisierung leicht mit der **Methode von Fermat** erfolgen. Diese Methode beruht auf folgender Idee: Können wir n als Differenz zweier Quadrate schreiben, $n = a^2 - b^2$, so ist n auch schon faktorisiert. Denn es ist ja $a^2 - b^2 = (a - b)(a + b)$! Kann man $n = pq$ *immer* in der Form $n = a^2 - b^2$ schreiben? – Ja, mit $a = \frac{p+q}{2}$ und $b = \frac{p-q}{2}$. Also braucht man nur dieses a und b zu finden, dann kann man daraus immer die Faktoren $p = a + b$ und $q = a - b$ berechnen. Fermat's Methode funktioniert nun so: Gegeben ist n, von dem wir wissen, dass es sich faktorisieren läßt. Aus $n = a^2 - b^2$ folgt $a^2 = n + b^2 \geq n$, daher $a \geq \sqrt{n}$. Für a kommen also nur die Zahlen $a_0 = \lceil \sqrt{n} \rceil$, $a_1 = a_0 + 1$, $a_2 = a_0 + 2$, ... in Frage. Wir testen also ob $\sqrt{a_0^2 - n}$ eine natürliche Zahl ist. Wenn ja, dann ist $a = a_0$ und $b = \sqrt{a_0^2 - n}$; wenn nein, probieren wir a_1 usw., bis $\sqrt{a_j^2 - n}$ eine natürliche Zahl ist. Diese ist dann gleich b und $a = a_j$.

Beispiel 5.42 Faktorisierung mit der Methode von Fermat
Versuchen Sie, $n = 221$ in seine Primfaktoren zu zerlegen.

Lösung zu 5.42 Berechne $\sqrt{221} = 14.87$; die kleinste natürliche Zahl $\geq \sqrt{n}$ ist $a_0 = 15$. Teste $\sqrt{15^2 - 221} = 2$. Das ist bereits eine natürliche Zahl, und damit schon das gesuchte b. Damit ist $a = 15, b = 2$ und $p = a - b = 13$, $q = a + b = 17$. Probe: $pq = 221$. ■

Der österreichische Briefbombenattentäter Franz Fuchs hat einen mit RSA verschlüsselten Brief an das Magazin Profil geschickt; dazu den (unverschlüsselten) triumphierenden Kommentar, dass man nun sicher die amerikanische NSA um Hilfe bitten müßte. Dabei hat er die mathematischen Fähigkeiten der Experten aber unterschätzt: Sein RSA-Modul n hatte zwar 243 Dezimalstellen, allerdings sind die Primfaktoren zu nahe beinandergelegen. Daher ist es innerhalb kürzester Zeit mithilfe von Fermat's Methode gelungen, die Primfaktoren zu finden und somit die Nachricht zu entschlüsseln. Auch Drucker der Firmen Canon und Fuji Xerox konnten noch im Jahr 2022 auf diese Weise angegriffen werden.

- p und q sollten nicht nahe beinander liegen.

Werden beide Primzahlen *zufällig* (und unabhängig) gewählt, so ist die Wahrscheinlichkeit für ein solches Paar nahe beinanderliegender Primzahlen vernachlässigbar.

5.5.3 $p \pm 1$ Faktorisierungsmethode

Um einen Primfaktor p einer Zahl n zu finden, kann man eine zu n teilerfremde Zahl x wählen (ist $\mathrm{ggT}(x, n) > 1$, so hat man schon einen Teiler gefunden und ist fertig) und eine Potenz k mit $x^k = 1 \pmod p$ und $x^k \neq 1 \pmod n$, denn dann ist $\mathrm{ggT}(x^k - 1, n)$ ein echter Teiler von n. Denn die erste Bedingung sagt ja, dass $x - 1$ ein Vielfaches des Teilers p ist und die zweite Bedingung stellt sicher, dass dieses Vielfache kein Vielfaches von n ist. Der Haken bei der Sache ist, dass wir p (noch) nicht kennen und somit nicht wissen wie wir k wählen müssen, um die erste Bedingung sicher zu stellen. Wir wissen aber, dass k ein Vielfaches von $\varphi(p) = p-1$ sein muss. Ist

$$p - 1 = p_1^{k_1} \cdots p_r^{k_r}$$

die Primfaktorzerlegung von $p - 1$, so muss

$$k = q \cdot p_1^{l_1} \cdots p_r^{l_r}$$

mit $l_j \geq k_j$ für $1 \leq j \leq r$ gelten. Wir wählen uns also eine Faktorbasis $p_1, \ldots, p_r$ aus Primzahlen (z.B. alle Primzahlen die kleiner als eine vorgegebene Schranke sind) und hoffen, dass die Primfaktoren von $p - 1$ darunter sind. Nun probieren wir Produkte von Potenzen der Primzahlen (z.B. wieder alle die kleiner als unsere Schranke sind) aus der Faktorbasis und berechnet $\mathrm{ggT}(x^k - 1, n)$. Dabei wird man schrittweise sowohl die Potenzen erhöhen als auch die Faktorbasis erweitern (also die gewählte Schranke erhöhen). Erreicht k irgendwann ein Vielfaches von n, so wird erfolglos abgebrochen und ein neues x gewählt. Das ist der $p-1$ **Faktorisierungsalgorithmus** von Pollard.

Beispiel 5.43 Faktorisieren Sie $n = 6589$ mit der $p - 1$ Methode.

Lösung zu 5.43 Wir wählen uns ein zufälliges x, z.B. $x = 2$ (damit wir weniger rechnen müssen und da x offensichtlich kein Teiler von n ist) und eine Schranke, z.B. $S = 5$. Dann ist $k = 2^2 * 3 * 5 = 60$ und

$$x^k = 2^{60} = 2212 \pmod{n}.$$

Der Euklid'sche Algorithmus liefert uns den Teiler $\text{ggT}(2211, 6589) = 11$. Hätten wir keinen Faktor gefunden, hätten wir die Schranke S erhöhen müssen. ■

Der Algorithmus scheitert, wenn es keinen Primfaktor p von n gibt, für den $p-1$ nur kleine Primfaktoren besitzt (deren Potenzen alle kleiner als die gewählte Schranke sind).

Wenn man den Algorithmus implementiert, benötigt man zunächst die Liste aller Primzahlen die kleiner als eine vorgegebene Schranke S sind. Dazu geht man der Reihe nach alle Zahllen $\leq S$ durch (beginnend bei 3 in Zweierschritten;-) und testet ob sie prim sind (dazu kommen wir im nächsten Abschnitt). Da es aufwendig ist, für jede Primzahl in der Faktorbasis jene Potenz zu bestimmen mit der sie gerade noch unter der gewählten Schranke S bleibt, nimmt man einfach das Produkt aller Zahlen $\leq \sqrt{S}$ und aller Primzahlen $\leq S$. Da wir ja sowieso nur hoffen die richtigen Faktoren dabei zu haben, verlieren wir dadurch nichts, gewinnen aber Effektivität. Ausserdem braucht man, wenn man die Schranke erhöht, k nicht vollkommen neu zu berechnen, sondern man mutlipliziert nur mit den Faktoren die neu dazu kommen.

Es gibt auch einen Algorithmus der anstelle von $\mathbb{Z}_p^*$ die speziellen Eigenschaften einer Untergruppe von $GF(p^2)$, die die Ordnung $p + 1$ hat, verwendet. In diesem Fall klappt der Algorithmus wenn $p + 1$ aus kleinen Primfaktoren besteht.

- Sowohl $p - 1$ als auch $p + 1$ (und analog für q) sollten große Primfaktoren besitzen.

Solche Primzahlen werden auch als **starke Primzahlen** bezeichnet.

Eine Verallgemeinerung des niederländischen Mathematikers Hendrik Lenstra (*1949) vermeidet die Nachteile der $p\pm1$ Methode indem es elliptische Kurven über $\mathbb{Z}_p$ (siehe Abschnitt 7.3) verwendet. Die Ordnung der Gruppe hängt nun von den Kurvenparametern ab und durch Variation dieser Parameter kann man die Wahrscheinlichkeit erhöhen, dass die Gruppenordnung ein Produkt kleiner Primfaktoren ist (siehe Abschnitt 7.3.1 für weitere Details). Moderne Siebverfahren (zu denen wir noch kommen) sind sogar noch effektiver. Dadurch relativieren sich die Vorteile starker Primzahlen und bei entsprechend großen Primzahlen ist der erhöhte Aufwand bei der Erzeugung nicht mehr sinnvoll [RS01]. Im Standard [FIPS186-5] werden weiterhin starke Primzahlen gefordert. Wie man solche findet, besprechen wir am Ende von Abschnitt 5.6.

5.5.4 Pollard Rho

In Abschnitt 5.3.3 haben wir die Pollard Rho-Mehode kennen gelernt. Diese kann auch adaptiert werden um Zahlen zu faktorisieren. Als Funktion f hat sich

$$f(x) = (x^2 + 2) \pmod{n}$$

als gut geeignet herausgestellt. Dann wählen wir uns einen Startwert x_0 und berechen die Folgen

$$x_{j+1} = f(x_j), \qquad y_{j+1} = f(f(y_j)).$$

Die Kollissionsbedingung ist

$$\text{ggT}(x_j - y_j, n) > 1.$$

Beispiel 5.44 Faktorisieren Sie $n = 6589$ mit der Pollard-Rho Methode.

Lösung zu 5.44 Wir wählen uns ein zufälliges x_0, z.B. $x_0 = y_0 = 4$ (damit wir weniger rechnen müssen und da x_0 offensichtlich kein Teiler von n ist). Dann ist $x_1 = 18$ und $y_1 = 326$. Der Euklid'sche Algorithmus liefert uns nach dem ersten Schritt den Teiler $\text{ggT}(308, 6589) = 11$. ■

In der Praxis ist das Berechnen des ggT aufwendiger als die Iteration. Deshalb könnte man nach jeweils k Iterationen das Produkt der entstehenden Differenzen $(x_j - y_j)(x_{j+1} - y_{j+1}) \cdots (x_{j+k} - y_{j+k})$ testen. Wenn man Pech hat, dann haben sich so viele Faktoren zusammengesammelt, dass man ein Vielfaches von n erreicht hat, aber dieser Fall ist eher unwahrscheinlich.

5.5.5 Faktorisierung mit dem quadratischen Sieb

Das quadratische Sieb kann als eine Weiterentwicklung der Fermat'schen Methode angesehen werden. Ziel der Fermat'schen Methode war es, n als Differenz zweier Quadrate zu schreiben. Dazu haben wir die Folge

$$f(j) = (j + m)^2 - n, \qquad m = \lceil \sqrt{n} \rceil,$$

betrachtet und j so lange erhöht bis das Ergebnis eine Quadratzahl war. Versuchen wir das mit $n = 6589$, so klappt das erst bei $j = 223$. In der Tat geht es aber deutlich schneller. Dazu betrachten wir

$$\begin{aligned} f(-2) &= -189 = -1 \cdot 3^3 \cdot 7, \\ f(-1) &= -28 = -1 \cdot 2^2 \cdot 7, \\ f(0) &= 135 = 3^3 \cdot 5, \\ f(1) &= 300 = 2^2 \cdot 3 \cdot 5^2, \\ f(2) &= 467. \end{aligned}$$

Keine dieser fünf Zahlen ist eine Quadratzahl, aber, wenn wir uns die Primfaktorzerlegung etwas genauer ansehen, dann erkennen wir dass

$$f(-2)f(-1)f(1) = 2^4 \cdot 3^4 \cdot 5^2 \cdot 7^2 = (2^2 \cdot 3^2 \cdot 5 \cdot 7)^2 = 1260^2$$

eine Quadratzahl ist. Also gilt

$$((m-2)(m-1)(m+1))^2 - 1260^2 = 4131^2 - 1260^2 = 0 \pmod{n}$$

und

$$\mathrm{ggT}(4131 - 1260, n) = 11, \qquad \mathrm{ggT}(4131 + 1260, n) = 599$$

liefern zwei Teiler von n.

Die Strategie ist somit klar: Wir berechnen $f(j)$ und versuchen ein Produkt zu finden, das eine Quadratzahl ergibt. Dazu müssen wir aber $f(j)$ faktorisieren. Dazu wählt man sich wieder eine Faktorbasis B und nur jene $f(j)$ die B-glatt sind, sind für uns brauchbar. Hat die Faktorbasis $B = \{p_0 = -1, p_1 = 2, \ldots, p_n\}$ insgesamt $n+1$ Elemente, so müssen so lange B-glatte $f(j) = p_0^{k_{j0}} \cdots p_n^{k_{jn}}$ gesucht werden, bis eine Linearkombination der Exponenten $k_j = (k_{j0}, \ldots, k_{jn})$ den Nullvektor in $\mathbb{Z}_2^{n+1}$ ergibt (die $k_{ji} \pmod 2$ bilden die Koeffizienten eines homogenen Gleichungssystem in $\mathbb{Z}_2$ von dem wir eine nichttriviale Lösung in $\mathbb{Z}_2$ suchen — Gauß-Algorithmus).

Das Problem dabei ist, dass man sehr viel Zeit mit Probedivisionen verschwendet, die am Ende nicht aufgehen. Hier kommt nun die Idee, die Zahlen bei denen die Probedivision aufgeht vorher auszusieben. Wir fragen uns also, wann $f(j) = 0 \pmod p$ ist? Da f ein quadratisches Polynom ist, gibt es im Körper $\mathbb{Z}_p^*$ maximal zwei Nullstellen die man durch Wurzelziehen in $\mathbb{Z}_p^*$ bestimmen kann (siehe Abschnitt 7.2). Dann gilt $f(j) = 0 \pmod p$ falls $j = -m \pm \sqrt{n} \pmod p$. Das liefert einen effektiven Test mit dem man vorweg die Zahlen aussieben kann für die die Faktorisierung per Probedivision sinnvoll ist. Insbesondere sind nur Primzahlen p_k die quadratische Reste modulo n sind für die Faktorbasis relevant (also Primzahlen mit $\left(\frac{p_k}{n}\right) = 1$; im Fall $\left(\frac{p_k}{n}\right) = 0$ hätten wir bereits einen Faktor).

In unserem Beispiel ist $\sqrt{n} = 1 \in \mathbb{Z}_3$ und somit ergeben sich die j, für die $f(j)$ durch 3 teilbar ist, als $-m \pm 1 + 3l$, $l \in \mathbb{Z}$. Analog gilt $\sqrt{n} = 2 \in \mathbb{Z}_5$, $\sqrt{n} = 3 \in \mathbb{Z}_7$ und damit sieht unser Sieb wie folgt aus:

j	$\ldots$	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	$\ldots$
$p_1 = 2$		$\star$		$\star$		$\star$		$\star$		$\star$		$\star$		
$p_2 = 3$		$\star$		$\star$	$\star$		$\star$	$\star$		$\star$	$\star$		$\star$	
$p_3 = 5$		$\star$	$\star$				$\star$	$\star$				$\star$	$\star$	
$p_4 = 7$					$\star$	$\star$						$\star$	$\star$	

Die ausgesiebten Zahlen j, die in einer der beiden Listen für p_l auftauchen, für die also $f(j)$ durch p_l teilbar ist, sind mit $\star$ markiert. Zusätzlich kann man beim Erstellen noch für jede Spalte die Summe über die (vorberechneten) Werte von $\log(p_k)$ bilden. Gibt es keine mehrfachen Primfaktoren, so muss die Summe gleich

$\log(|f(j)|)$ ergeben. Da diese Annahme für kleine Primfaktoren öfter nicht gilt, arbeitet man mit einer Schranke. Nur bei Zahlen die diese Schranke erreichen ist es sinnvoll die Faktorisierung mittels Probedivision zu versuchen.

Das Problem bei dem Verfahren ist, dass man das Intervall der j-Werte so lange erhöhen muss, bis man genug B-glatte Zahlen $f(j)$ gefunden hat, sodass man aus dem Produkt eine Quadratzahl erhält. Umso größer aber $|j|$ wird, um so größer wird auch $f(j)$ und die Wahrscheinlichkeit, dass $f(j)$ B-glatt ist sinkt. Also muss man B erhöhen und dann benötigt man wiederum noch mehr B-glatte Zahlen! Der Ausweg aus diesem Dilemma ist das **Multipolynomiale Quadratische Sieb** (**MPQS**) bei dem man das einzelne Polynom x^2-n durch mehrere Polynome der Form $ax^2 + 2bx + c$ mit Diskriminante $b^2 - ac = n$ ersetzt. Wir wollen hier aber nicht weiter darauf eingehen (siehe [For15]). Bis ca. 110 Dezimalstellen ist es der derzeit schnellste Algorithmus. Dieser wird angeworfen, nachdem man mittels Probedivision bzw. der $p \pm 1$-Methode und der Elliptischen-Kurven-Methode kleine Primfaktoren erledigt hat. Ausserdem verwendet man Primzahltests (Abschnitt 5.6) bevor man sinnlos Rechenzeit verschwendet. Im Jahr 1994 gelang es, mit Hilfe des MPQS die 129-stellige Zahl RSA-129 zu faktorisieren.

Das schnellste heute bekannte Verfahren zur Faktorisierung von Zahlen mit mehr als 110 Dezimalstellen ist das **Zahlkörpersieb** ((General) Number Field Sieve, **GNFS**). Es ist eine Verallgemeinerung des Quadratischen Siebs bei dem mit Polynomringen (Körpererweiterungen) gearbeitet wird, in denen glatte Zahlen häufiger auftreten und somit schneller gefunden werden können. Damit gelang 2005 die Faktorisierung von RSA-200. Es gibt auch noch das spezielle Zahlkörpersieb (**SNFS**), das für Zahlen der Form $r^k \pm s$ mit r und s klein, optimiert ist.

5.5.6 Shor Algorithmus

Zuletzt erwähnen wir noch einen Zusammenhang mit dem Auffinden der Ordnung eines beliebigen Elements. Wir wählen also eine zufällige Zahl $a \in \mathbb{Z}_n$. Ist $\mathrm{ggT}(a, n) \neq 1$, so haben wir einen Teiler von n gefunden und sind fertig. Wenn nicht, bestimmen wir die Ordnung k von $a \in \mathbb{Z}_n^*$, also die kleinste natürliche Zahl k mit

$$a^k = 1 + jn$$

für ein $j \in \mathbb{N}$. Ist nun weiters k gerade, so können wir diese Gleichung wie folgt faktorisieren:

$$jn = (a^{k/2} - 1)(a^{k/2} + 1).$$

Also muss entweder $(a^{k/2}-1)$ oder $(a^{k/2}+1)$ einen Teiler von n enthalten. Trotzdem kann es passieren, dass weder $\mathrm{ggT}(a^{k/2} - 1, n)$ noch $\mathrm{ggT}(a^{k/2} + 1, n)$ einen echten Teiler von n liefern.

Beispiel 5.45 Für $n = 15$ ist die Ordnung von $\mathbb{Z}_n^*$ gleich $\varphi(15) = (5-1)(3-1) = 8$ und die möglichen Ordnungen eines Elements sind nach dem Satz von Lagrange $2, 4, 8$. Somit haben alle Zahlen (ausser 1) gerade Ordnung. Mehr noch, man überprüft leicht, dass alle Zahlen $a = 2, 4, 7, 8, 11, 13$ ausser $a = 14$ obige Bedingung erfüllen und damit die Teiler von n liefern. Z.B. hat $a = 2$ die Ordnung $k = 4$ und es gilt $\mathrm{ggT}(a^{k/2} - 1, n) = \mathrm{ggT}(3, 15) = 3$, $\mathrm{ggT}(a^{k/2} + 1, n) = \mathrm{ggT}(5, 15) = 5$. Für $a = 14$ gilt $k = 2$ und $\mathrm{ggT}(a^{k/2} - 1, n) = \mathrm{ggT}(13, 15) = 1$, $\mathrm{ggT}(a^{k/2} + 1, n) = \mathrm{ggT}(15, 15) = 15$.

So wie im letzten Beispiel, stellt sich allgemein heraus, dass der Fall, in dem wir keinen Teiler bekommen, nicht sehr wahrscheinlich ist. Das sagt folgender Satz der auf Ideen des amerikanischen Informatikers Gary L. Miller basiert.

Satz 5.46 Ist n ungerade und keine Primzahlpotenz, so gilt für mindestens die Hälfte aller Zahlen in $\mathbb{Z}_n^*$, dass

- die Ordnung k von a gerade ist und
- n die Zahl $a^{k/2} + 1$ nicht teilt.

Da der Fall, dass n die Zahl $a^{k/2} - 1$ teilt, nicht eintreten kann (denn dann wäre nicht k sondern $k/2$ die Ordnung), liefert sowohl jede Zahl $a \in \mathbb{Z}_n \setminus \mathbb{Z}_n^*$ wie auch jede Zahl $a \in \mathbb{Z}_n^*$ die die Bedingungen aus dem Satz erfüllt einen nichttrivialen Teiler von n.

Der hier beschriebene Algorithmus findet also nach wenigen Versuchen mit hoher Wahrscheinlichkeit einen Faktor von n (es geht sogar noch besser, [Eke21]). Insbesondere sehen wir, dass ein effektiver Algorithmus zur Bestimmung der Ordnung eines Elements in $\mathbb{Z}_n^*$ auch einen effektiven Algorithmus zur Entschlüsselung von RSA liefert.

Wir haben hier in der Tat einen allgemeinen Faktorisierungsalgorithmus. Denn ist n gerade, so ist nichts zu tun und ob n eine Primzahlpotenz ist kann man feststellen indem man in $\mathbb{R}$ Wurzeln zieht und überprüft ob das Ergebnis ganzzahlig ist. Umgekehrt liefert ein Algorithmus zur Faktorisierung auch einen Algorithmus zur Bestimmung der Ordnung, wie wir in Satz 5.25 gesehen haben.

Das Auffinden der Ordnung kann auch als das Auffinden der Periode der Funktion

$$f : \mathbb{Z} \to \mathbb{Z}_n, \qquad j \mapsto a^j$$

betrachtet werden, denn es gilt ja $f(j+k) = f(j)$ mit $k = \mathrm{ord}(a)$. Perioden können mithilfe der Fouriertransformation gefunden werden und letztere kann effektiv von einem Quantencomputer berechnet werden. Das ist die Grundidee des **Shor Algorithmus** [Sho97a] zur Faktorisierung auf einem Quantencomputer. Wir werden darauf in Studienbrief 8 zurückkommen.

5.6 Primzahltests

Wie findet Alice nun zur Schlüsselerzeugung große Primzahlen, bzw. gibt es überhaupt ausreichend viele Primzahlen mit z.B. 512 Bit? Das ist essentiell, denn gäbe es zu wenige Primzahlen mit fester Länge, so könnte ein Angreifer einfach alle Primzahlen dieser Länge abspeichern und durchprobieren.

Wie wir wissen, gibt es kein *Bildungsgesetz*, das uns die Folge aller Primzahlen liefert. Man kann aber abschätzen, wie viele Primzahlen es kleiner oder gleich einer bestimmten Zahl gibt, und damit kann man auch abschätzen, wie viele Primzahlen es in einem bestimmten Intervall gibt:

Satz 5.47 (Primzahlsatz) Bezeichnet $\pi(n)$ die Anzahl aller Primzahlen, die kleiner oder gleich n sind. Dann gilt für großes n näherungsweise

$$\pi(n) \approx \frac{n}{\ln(n)}.$$

Obiges $\approx$ ist in folgendem Sinn zu verstehen: Es gilt

$$\pi(n) = \frac{n}{\ln(n)}(1 + o(1)),$$

wobei $o(1)$ für eine Folge $a(n)$ steht, die für $n \to \infty$ gegen 0 konvergiert. Mit anderen Worten, der Quotient $\pi(n)\frac{\ln(n)}{n} = 1 + o(1)$ konvergiert für $n \to \infty$ gegen 1. Man kann sogar $\frac{n}{\ln(n)} \leq \pi(n)$ für $n \geq 1$ und $\pi(n) \leq 1.25506\frac{n}{\ln(n)}$ für $n \geq 17$ zeigen.

Der Satz impliziert auch, dass für die n'te Primzahl $p_n \approx n\ln(n)$ gilt. Man kann sogar $n(\ln(n) + \ln(\ln(n)) - 1) < p_n < n(\ln(n) + \ln(\ln(n)))$ für $n \geq 6$ zeigen.

Beispiel 5.48 (→CAS) Primzahlsatz
Berechnen Sie für $n = 10, 100, 10^3, 10^4, 10^5$: a) $\pi(n)$ b) $\frac{n}{\ln(n)}$ c) $\pi(n)\frac{\ln(n)}{n}$

Lösung zu 5.48 Mit dem Computer erhält man:

n	$\pi(n)$	$\frac{n}{\ln(n)}$	$\pi(n)\frac{\ln(n)}{n}$
10	4	4.34294	0.921034
100	25	21.7147	1.15129
1000	168	144.765	1.1605
10000	1229	1085.74	1.13195
100000	9592	8685.89	1.10432

In der ersten Spalte steht also n, und daneben in der zweiten Spalte steht, wie viele Primzahlen es (exakt) bis n gibt. Es gibt also zum Beispiel bis $n = 10$ insgesamt $\pi(10) = 4$ Primzahlen. Daneben steht in der dritten Spalte der Wert, den uns die Näherungsfunktion für $\pi(10)$ liefert. Aus der vierten Spalte sieht man, dass diese Näherung umso besser wird, je größer n ist. Denn der Quotient aus $\pi(n)$ und $\frac{n}{\ln(n)}$ liegt mit wachsendem n in der Tat immer näher bei 1. ■

Wir möchten nun eine Vorstellung davon bekommen, wie viele Primzahlen es mit Länge 512 Bit gibt.

Wie viele Primzahlen mit 3 Bit gibt es? Die kleinste Zahl mit 3 Bit ist $(100)_2 = 4$, die größte ist $(111)_2 = 7$. Das heißt, dass alle 3-Bit-Zahlen Zahlen größergleich $4 = 2^2$ und kleiner als $8 = 2^3$ sind. Die gesuchte Anzahl der Primzahlen mit 3 Bit ist daher $\pi(2^3) - \pi(2^2) = 2$.

Alle Zahlen der Länge l Bit sind größer oder gleich 2^{l-1} und kleiner 2^l, insbesondere liegen daher alle Primzahlen der Länge l Bit zwischen 2^{l-1} und 2^l. Aus dem Primzahlsatz folgt daher, dass es näherungsweise

$$\pi(2^l) - \pi(2^{l-1}) \approx \frac{2^l(l-2)}{l(l-1)\ln(4)}.$$

Primzahlen mit l Bit gibt (das Nachrechnen dieser Formel ist eine Übungsaufgabe :-). Man beachte, dass diese Zahl mit l exponentiell wächst.

Beispiel 5.49 Anzahl der Primzahlen bestimmter Länge
Wie viele Primzahlen mit 512 Bit gibt es näherungsweise?

Lösung zu 5.49 Für $l = 512$ erhalten wir

$$\frac{2^l(l-2)}{l(l-1)\ln(4)} = 1.9 \cdot 10^{151}$$

Es gibt also ungefähr $2 \cdot 10^{151}$ Primzahlen mit $l = 512$ Bit. Diese Zahl ist größer als die Anzahl der Atome im bekannten Universum. ■

Wir brauchen uns also keine Sorgen machen, dass uns irgendwann die Primzahlen ausgehen. Und auch alle Angriffe, die auf dem Durchprobieren aller Primzahlen beruhen, sind von vornherein zum Scheitern verurteilt.

Nachdem wir nun wissen, dass es genügend Primzahlen der Länge 512 Bit gibt, wie finden wir nun eine? Wenn wir aus allen Zahlen der Länge l Bit *zufällig* eine herausgreifen, so ist die Wahrscheinlichkeit dafür, dass es eine Primzahl ist, gleich

$$\frac{\pi(2^l) - \pi(2^{l-1})}{2^l - 2^{l-1}} \approx \frac{l-2}{l \cdot (l-1) \cdot \ln(2)}$$

(= Anzahl der Primzahlen der Länge l geteilt durch die Anzahl aller Zahlen der Länge l = günstige Fälle durch mögliche Fälle). Man beachte, dass diese Zahl mit l kleiner wird.

Beispiel 5.50 Wahrscheinlichkeit, zufällig eine Primzahl zu treffen
Wie groß ist die Wahrscheinlichkeit, bei zufälliger Wahl einer 512-Bit-Zahl eine Primzahl zu treffen?

Lösung zu 5.50 Für $l = 512$ erhalten wir

$$\frac{l-2}{l \cdot (l-1) \cdot \ln(2)} = 0.0028.$$

Wählen wir nur ungerade Zahlen, so verdoppeln sich unsere Erfolgschancen.

Konkret für 512 Bit finden wir also mit Wahrscheinlichkeit $p = 0.0056$ eine Primzahl und der Erwartungswert für die Anzahl bis zum ersten Erfolg (Binomialverteilung) ist $\frac{1}{p} = 178$. Allgemein gilt $\frac{1}{p} \approx \ln(2)l$, die erwartete Anzahl an Versuchen wächst also linear mit l.

Man muss also nur genügend Zahlen probieren, dann trifft man früher oder später in praktisch sinnvoller Zeit auf eine Primzahl. ■

Man geht in der Praxis also bei der Wahl der Primfaktoren so vor, dass man zufällig Zahlen in der gewünschten Größenordnung wählt und daraufhin testet, ob es sich um eine Primzahl handelt (**Primzahltest, primality testing**). Das letzte Problem ist nun also der Test, ob eine Zahl eine Primzahl ist, oder nicht. Zu versuchen, die Zahl zu faktorisieren kommt natürlich nicht in Frage.

Man versucht daher Primzahlen anhand ihrer Eigenschaften zu erkennen. Die einfachste Idee ist es, den Satz von Fermat zu verwenden. Beim **Fermat-Test** überprüft man also für die zu untersuchende Zahl n die Beziehung

$$a^{n-1} = 1 \pmod{n}.$$

Scheitert der Test für ein $1 < a < n$, so ist n sicher keine Primzahl. Eine solche Zahl a wird als Fermat-Zeuge für (die Zusammengesetztheit von) n bezeichnet. Leider ist nicht jedes a ein Fermat-Zeuge. Z.B. besteht $n = 9$ den Test für $a = 8$ und $n = 15$ für $a = 4, 11, 14$. Falls a und n einen gemeinsamen Teiler haben, so scheitert der Test immer und alle $a \notin \mathbb{Z}_n^*$ sind somit Fermat-Zeugen.

Sei $a = t \cdot b$ und $n = t \cdot m$; dann folgt aus $a^{n-1} = 1 \pmod{n}$ der Widerspruch $m = m \cdot a^{n-1} = n \cdot b \cdot a^{n-2} = 0 \pmod{n}$.

Im Prinzip können wir damit Primzahlen identifizieren, indem wir alle $1 < a < n-1$ durchprobieren. Bestehen a und b beide den Test, so gilt das auch für das Produkt $a \cdot b$ (da ja $(a \cdot b)^{n-1} = a^{n-1}b^{n-1}$). Es reicht also alle Primzahlen kleiner n zu testen; das sind aber, wie wir gesehen haben, immer noch zu viele! Da man in der Praxis viele Zahlen testen muss, ist es außerdem wichtig, dass dieser Test schnell geht. Deshalb testet man nur einige wenige a und man spricht von einem *statistischen Test*. Diese sagen einem zwar nur mit an Sicherheit grenzender Wahrscheinlichkeit, ob es sich um eine Primzahl handelt – für die Praxis ist das aber gut genug.

Unser Fermat-Test eignet sich dafür allerdings nicht: die Zahl $561 = 3 \cdot 11 \cdot 17$ besteht den Test für alle zu n teilerfremden a. Es gibt also ausser $\mathbb{Z}_n \setminus \mathbb{Z}_n^*$ keine weiteren Fermat-Zeugen. Eine solche Zahl wird als **Carmichael-Zahl** bezeichnet. Für 561 scheitert unser Primzahltest in über $\frac{\varphi(n)}{n-2} = 57\%$ der Fälle bei zufällig gewählten a. Carmichael-Zahlen sind zwar relativ selten (bis 10^{15} gibt es nur 105 212), aber es gibt unendlich viele. Für größere Carmichael-Zahlen steigt der

Wert weiter und damit gibt es Zahlen die von diesem Test auch bei mehfacher Anwendung mit verschiedenen a nicht zuverlässig als zusammengesetzt erkannt werden.

Eine Verbesserung erhält man, wenn man $b = a^{(p-1)/2}$ betrachtet. Dann gilt $b^2 = 1 \pmod p$ und da diese Gleichung im Körper $\mathbb{Z}_p$ nur die beiden Lösungen ± 1 besitzt, muss für eine Primzahl p immer $a^{(p-1)/2} \in \{\pm 1\}$ gelten. Wenn man diese Idee etwas weiterspinnt, erhält man den Primzahltest von Miller und Rabin der auf folgender Tatsache basiert:

Satz 5.51 (Miller-Rabin-Test) Sei n eine ungerade Zahl der Form $n = 2^k m+1$ mit m ungerade und $k \geq 1$. Eine Zahl $a \in \mathbb{Z}_n \backslash \{0\}$ heißt Miller–Rabin-Zeuge für (die Zusammengesetztheit von) n, wenn

$$a^m \neq 1 \pmod n \quad \text{und} \quad a^{2^i m} \neq -1 \pmod n \text{ für alle } i \in \{0, \ldots, k-1\}.$$

Es gilt:

a) Ist n Primzahl, so gibt es keine Miller–Rabin-Zeugen.
b) Ist eine ungerade Zahl $n > 9$ keine Primzahl, so sind mindestens $\frac{3}{4}$ aller $a \in \mathbb{Z}_n^*$ und alle $a \notin \mathbb{Z}_n^*$ Miller–Rabin-Zeugen.

Warum? Ist n prim, dann muss in der Folge a^m, a^{2m}, a^{4m},…,$a^{2^k m} = a^{n-1} \pmod n$ spätestens an der letzten Stelle eine 1 stehen. Da in dieser Folge sukzessive quadriert wird, muss an der Stelle vor der ersten 1 der Wert -1 stehen. Also besteht die Folge entweder nur aus dem Wert 1 (falls $a^m = 1$) oder sie enthält den Wert -1 und damit kann es für Primzahlen keine Miller–Rabin-Zeugen geben.

Da jeder Miller–Rabin-Zeuge auch Fermat-Zeuge ist, sind alle $a \notin \mathbb{Z}_n^*$ Miller–Rabin-Zeugen, wie wir uns schon vorher überlegt haben. Der Fall, bei dem a und n teilerfremd sind, ist komplizierter.

Daraus ergibt sich der folgende **statistische Primzahltest nach Miller und Rabin** für eine ungerade Zahl n:

- Schreibe n in der Form $n = 2^k m + 1$ mit ungeradem m.
- Wähle ein $a \in \{2, 3, \ldots, n-2\}$. (Denn $a = 1$ bzw. $a = -1 = n - 1 \,(\mathrm{mod}\, n)$ wird immer $a^m = 1 \,(\mathrm{mod}\, n)$ oder $a^m = -1 \,(\mathrm{mod}\, n)$ liefern, also nie Zeuge sein.)
- Teste, ob $\mathrm{ggT}(a, n) > 1$ ist. Wenn ja, ist n zusammengesetzt und es kann abgebrochen werden.
- Berechne a^m. Ist $a^m \in \{\pm 1\}$ (kein Zeuge) wähle eine neue Basis. Ansonsten quadriere a^m maximal $k-1$ Mal. Wird irgendwann -1 erreicht (kein Zeuge), wähle eine neue Basis. Ansonsten ist n zusammengesetzt und es kann abgebrochen werden.

- Wenn genügend viele Basen a getestet wurden und kein Zeuge darunter war, so ist n mit hoher Wahrscheinlichkeit prim.

Der Test ob $\text{ggT}(a,n) > 1$ kann auch übersprungen werden, da in diesem Fall ja auch der nächste Schritt n als zusammengesetzt erkennt. Außerdem wählt man in der ersten Runde oft $a = 2$, da dann der Test besonders schnell ausgeführt werden kann (und damit schon die meisten Zahlen als zusammengesetzt erkannt werden). Auch ist es sinnvoll vor dem Test mittels Probedivision auf kleine Primteiler zu testen.

Beispiel 5.52 Primzahltest nach Miller und Rabin
Testen Sie mit dem Primzahltest, ob die Zahl n eine Primzahl ist:
a) $n = 41$ b) $n = 145$ c) $n = 341$

Lösung zu 5.52

a) $n = 2^3 \cdot 5 + 1$, also $m = 5$ und $k = 3$. Wir müssen also $a^m, a^{2m}, a^{2^2 m} = a^{4m}$ testen für Zufallszahlen a aus $\{2, 3, \ldots, 39\}$. Wähle z.B. für a die Werte $3, 4, 9, 16$:
$3^5 = 38 \pmod{41}$, $3^{10} = 9 \pmod{41}$, $3^{20} = 40 = -1 (\text{mod}\, 41)$, somit ist 3 kein Zeuge.
$4^5 = -1 \pmod{41}$, also wissen wir schon, dass auch 4 kein Zeuge ist.
$9^5 = 9 \pmod{41}$, $9^{10} = -1 \pmod{41}$ also ist auch 9 kein Zeuge ist.
$16^5 = 1 \pmod{41}$, somit ist auch 16 kein Zeuge.
Es spricht also der Test bisher nicht dagegen, dass 41 eine Primzahl ist.

b) $n = 2^4 \cdot 9 + 1$, also $m = 9$ und $k = 4$. Wir müssen also $a^m, a^{2m}, a^{2^2 m} = a^{4m}$, $a^{2^3 m} = a^{8m}$ testen für Zufallszahlen a aus $\{2, 3, \ldots, 143\}$. Wähle z.B. für a die Werte $2, 5, 8$:
$2^9 = 77 (\text{mod}\, 145)$, $2^{18} = 129 \pmod{145}$, $2^{36} = 111 \pmod{145}$, $2^{72} = 141 \pmod{145}$. Da sich für alle Testpotenzen mit Basis 2 ein Ergebnis ungleich ± 1 ergab, ist $a = 2$ Zeuge dafür, dass $n = 145$ keine Primzahl ist und wir können den Test schon beenden.

c) $n = 2^2 \cdot 85 + 1$, also $m = 85$ und $k = 2$. Wir müssen also a^m und a^{2m} testen für Zufallszahlen a aus $\{2, 3, \ldots, 339\}$. Wähle z.B. für a die Werte $2, 3, 8$:
$2^{85} = 32 \pmod{341}$, $2^{170} = 1 \pmod{341}$, somit ist 2 ein Zeuge, dass 341 keine Primzahl ist. Beachte, dass in diesem Fall $2^{n-1} = 1 \pmod{341}$ gilt. ■

Wenn die Entscheidung „keine Primzahl“ fällt, so ist sie immer richtig. Die Entscheidung „Primzahl“ kann aber falsch sein, wenn nicht ausreichend viele Basen getestet wurden. Denn es kann passieren, dass n keine Primzahl ist, aber *keine* der zufällig gewählten a_i Miller–Rabin-Zeugen sind. Dann liefert der Test die (falsche) Antwort: „Primzahl“. Aus unserem Satz folgt, dass höchstens $\frac{\varphi(n)}{4}$ keine Zeugen sind, dass also $\frac{\varphi(n)}{4} + 1$ Basen genügen um diese Entscheidung mit Sicherheit zu treffen, aber bei großen Zahlen ist es in der Praxis nicht möglich, so viele Tests

durchzuführen und man begnügt sich daher mit einer gewissen Fehlerwahrscheinlichkeit (es gibt ja auch eine gewisse Wahrscheinlichkeit, dass ein Computer falsch rechnet — defekter Speicherchip, etc.). Die Wahrscheinlichkeit für diese Fehlentscheidung ist für r getestete Zahlen $a_1, \ldots, a_r$ kleiner oder gleich $(\frac{1}{4})^r$. Die Antwort „Primzahl“ wird also immer zuverlässiger, je mehr Zahlen man aus $\mathbb{Z}_n$ testet.

Die Wahrscheinlichkeit, im Fall keiner Primzahl ein „schlechtes“ a_i zu erwischen, ist $\leq \frac{1}{4}$, denn nach Satz 5.51 ist maximal ein Viertel aller a_i „schlecht“ (also kein Zeuge). Damit ist die Wahrscheinlichkeit, *alle* r zufällig gewählten Zahlen $a_1, \ldots, a_r$ schlecht zu erwischen (mit Zurücklegen), $\leq (\frac{1}{4})^r$. In der Praxis interessant ist jedoch die Wahrscheinlichkeit, dass eine Zahl r Tests positiv überstanden hat, jedoch keine Primzahl ist. Diese kann wegen $0 \leq P(T|\overline{P}) \leq (\frac{1}{4})^r$ und $P(T|P) = 1$ abgeschätzt werden durch (Bayes'sche Formel)

$$P(\overline{P}|T) = \frac{P(T|\overline{P}) \cdot P(\overline{P})}{P(T|\overline{P}) \cdot P(\overline{P}) + P(T|P) \cdot P(P)} \leq \frac{4^{-r} \cdot P(\overline{P})}{4^{-r} \cdot P(\overline{P}) + 1 \cdot P(P)} \leq \frac{1}{4^r P(P)}$$

wobei T bedeutet: „r Tests überstanden“ und P bedeutet: „Die zufällig gewählte Zahl ist eine Primzahl“. Beispiel: Bei zufälliger Wahl einer ungeraden 512-Bit Zahl ist $P(P) \approx 0.0056$. Damit ergibt sich hier für die Wahrscheinlichkeit, dass eine Zahl $r = 20$ Tests positiv übersteht, obwohl sie keine Primzahl ist, $\leq 1.6 \cdot 10^{-10}$.

In der Regel ist die Anzahl der Nicht-Zeugen wesentlich kleiner. Man kann diese Wahrscheinlichkeiten auch noch genauer abschätzten und ein guter Überblick dazu findet sich in [MOV96]. Zahlen bei denen die Maximalzahl der Nicht-Zeugen $\frac{\varphi(n)}{4}$ angenommen wird, werden auch als starke Pseudoprimzahlen bezeichnet und sind relativ selten. Die vierstelligen starken Pseudoprimzahlen sind übrigens $\frac{\varphi(15)}{4} = 2$, $\frac{\varphi(91)}{4} = 18$, $\frac{\varphi(703)}{4} = 162$, $\frac{\varphi(1891)}{4} = 450$ und $\frac{\varphi(8911)}{4} = 1782$.

Im Jahr 2002 wurde ein Algorithmus veröffentlicht, der es erlaubt, in polynomialer Zeit *mit Sicherheit* festzustellen (man spricht dann von einem *deterministischen* Test) ob eine Zahl eine Primzahl ist oder nicht (**AKS-Primzahltest**). Die Laufzeit des Tests ist aber für die Praxis meist zu hoch, daher werden nach wie vor *probabilistische* Tests wie der Miller-Rabin-Test verwendet.

Es gibt übrigens auch statistische Tests die mit Sicherheit Primzahlen erkennen. Typischerweise benötigen diese aber Zusatzinformationen über die zu testende Zahl n. Z.B. kann man Primzahlen daran erkennen, dass es in $\mathbb{Z}_p^*$ Generatoren der Ordnung $p-1$ gibt. Ist n keine Primzahl, so gibt es für $a \in \mathbb{Z}_n \setminus \mathbb{Z}_n^*$ überhaupt kein $j \in \mathbb{N}$ mit $a^j = 1$ und auch für $a \in \mathbb{Z}_n^*$ ist die maximale Ordnung $\lambda(n) < n - 1$. Sind die Primfaktoren q_j von $n - 1$ bekannt, so können wir leicht testen, ob die Ordnung von a gleich $n - 1$ ist: Ist $a^n \neq 1$ so haben wir $a \in \mathbb{Z}_n \setminus \mathbb{Z}_n^*$ erwischt und n ist zusammengesetzt (Fermattest). Anderenfalls testen wir ob $a^{n/q_j} \neq 1$ für alle Primteiler von $n - 1$ gilt. Wenn ja, dann ist a ein Generator der Ordnung $n - 1$ (Satz 5.25) und damit n prim. Wenn nein, muss ein neues a gewählt werden. Die Wahrscheinlichkeit im Fall einer Primzahl einen Generator zu erwischen ist $\frac{\varphi(p-1)}{p-1}$ und der Erwartungswert für die Anzahl der Versuche bis zum ersten Treffer (Binomialverteilung) ist $\frac{p-1}{\varphi(p-1)} < 6 \ln(\ln(p))$. Tritt also dieses Ereignis nicht innerhalb der erwarteten Anzahl ein, so sollte abgebrochen werden und die Zahl ist wahrscheinlich zusammengesetzt. Wird ein Generator gefunden, so ist die Zahl sicher prim.

Diese Ideen können weiterentwickelt werden und es reicht wenn man nur einen Teil der Primfaktoren kennt (**Satz von Pocklington**). Einen Überblick findet man in [MOV96]. Damit kann man dann auch Zufallsprimzahlen erzeugen die beweisbar prim sind (**Maurer-Algorithmus**, [Mau95]).

Zum Schluß machen wir noch ein paar Anmerkungen, wie für kryptographische Verfahren geeignete Primzahlen erzeugt werden können. Zunächst ist festzuhalten, dass der entscheidende Faktor für die Sicherheit, die Güte der verwendeten Zufallszahlen ist.

Wenn Sie einen Pseudozufallsgenerator mit der Uhrzeit initialisieren, dann haben Sie schon verloren (kennt Eve die ungefähre Uhrzeit zu der ihr Schlüssel erzeugt wurde, kann die Brute-Force Suche schon beginnen). Man kann auch im Internet öffentliche RSA-Schlüssel mit gemeinsamen Teilern suchen um zu sehen, dass diese Vorgaben in der Praxis nicht immer berücksichtigt werden. Von inoffiziellen *schnellen* Methoden Primzahlen zu erzeugen ist auch abzuraten (vgl. **ROCA-Verwundbarkeit**, [Nem+17]).

Um eine Primzahl passender Größe zu erzeugen, starten wir mit einer Zufallszahl in der gewünschten Größe, stellen sicher dass sie ungerade ist und erhöhen dann solange um 2 bis die Zahl von unserem Primzahltest als Primzahl erkannt wird. Wollen wir sicherstellen, dass auch $p-1$ einen großen Primfaktor q enthält, so erzeugen wir zuerst q und erhöhen dann k in

$$2kq+1$$

so lange bis wir auf eine Primzahl treffen. Dass dieses Verfahren immer funktioniert garantiert uns der

Satz 5.53 (Dirichlet'scher Primzahlsatz) Es sei $a \in \mathbb{Z}_m^*$. Bezeichnet $\pi(n,m,a)$ die Anzahl aller Primzahlen von der Form $a+km$, $k \in \mathbb{N}$, die kleiner oder gleich n sind. Dann gilt für großes n näherungsweise

$$\pi(n,q,a) \approx \frac{n}{\varphi(m)\ln(n)}.$$

Das bedeutet also, dass die Primzahlen gleichmäßig über die $\varphi(m)$ Kongruenzklassen in $\mathbb{Z}_m^*$ verteilt sind.

Ein Verfahren, dass diese Idee weiterentwickelt, ist der Algorithmus von Gorden [Gor84] zur Erzeugung starker Primzahlen:

- Zu vorgegebenen Primzahlen r, s werden Primzahlten $r > 2t$ und $p > 2rs$ erzeugt, so dass $p+1$, $p-1$, $r-1$ Primfaktoren t, r, s enthalten.
- Man startet mit zwei Primzahlen s und t in der gewüschten Größenordnung.

- Nun wählt man einen Startwert $i \geq 1$ und erhöht i solange bis $r_i = 2it + 1$ prim ist. Diese Primzahl nennen wir r (das wird ein Primfaktor von $p - 1$).

- Nun bilden wir $p_0 = 2s(s^{r-2} \bmod r) - 1$, wählen wieder einen Startwert $j \geq 1$ und erhöht j solange bis $p_j = p_0 + 2rsj$ prim ist. Diese Primzahl ist die gesuchte starke Primzahl p.

Auf jeden Fall ist wegen $r = 2ti + 1$ die Zahl t ein Primfaktor von $r - 1$. Nach Konstruktion gilt $p_0 = 1 \pmod r$ und $p_0 = -1 \pmod s$. Also ist $p - 1 = p_0 + 2rsj = 0 \pmod r$ und somit r ein Primfaktor von $p - 1$. Analog ist $p + 1 = p_0 + 2rsj = 0 \pmod s$ und somit s ein Primfaktor von $p + 1$.

Für weitere Algorithmen sei auf [FIPS186-5] verwiesen. Insbesondere werden dort auch Algorithmen beschrieben mit denen man zufällige Primzahlen erzeugen kann die beweisbar prim sind (und nicht nur wahrscheinliche Primzahlen, wie sie bei Verwendung eines statistischen Primzahltests erzeugt werden).

Falls Sie sich wundern warum in [FIPS186-5] Hashfunktionen zum Erzeugen von Zufallsprimzahlen verwendet werden: Verwendet man die Zufallszahlen nicht direkt, sondern deren Hashwert um eine Primzahl zu erzeugen, so kann man die Zufallszahl, die zur gewählten Primzahl geführt hat, bekannt geben und damit beweisen, dass die Primzahl wirklich zufällig (und nicht absichtlich schwach) gewählt wurde. Es ist nämlich möglich **gezinkte Primzahlen** zu erzeugen bei denen eine geheime Zusatzinformation die Lösung des DLP erleichtert [Fri+17].

5.7 Kontrollfragen

Fragen zu Abschnitt 5.1: Public-Key-Verfahren

Erklären Sie folgende Begriffe und überprüfen Sie Ihre Antwort mit dem Skriptum: symmetrisches/asymmetrisches Verfahren, private key, public key, No-Key-Protokoll, Diffie–Hellman-Schlüsselvereinbarung, Einwegfunktion, diskrete Exponentialfunktion, Elgamal Verfahren.

1. Was versteht man unter einem symmetrischen, was unter einem asymmetrischen Verschlüsselungsverfahren?

   (Lösung zu Kontrollfrage 1)

2. Wie funktioniert das Ver- und Entschlüsseln einer Nachricht bei einem asymmetrisches Verfahren?

   (Lösung zu Kontrollfrage 2)

3. Was ist eine Einweg-Falltürfunktion?

   (Lösung zu Kontrollfrage 3)

4. a) Welches Problem löst das No-Key-Protokoll?
   b) Wie kann es anschaulich beschrieben werden?
   c) Was ist dafür mathematisch notwendig?
   d) Nennen Sie Nachteile dieses Protokolls.

   (Lösung zu Kontrollfrage 4)

5. Warum kommen das One-Time-Pad oder DES nicht für das No-Key-Protokoll in Frage?

   (Lösung zu Kontrollfrage 5)

6. a) Wie hängt die Diffie–Hellman-Schlüsselvereinbarung mit dem No-Key-Protokoll zusammen?
   b) Was ist mathematisch für die Diffie–Hellman-Schlüsselvereinbarung notwendig?
   c) Warum ist die diskrete Exponentialfunktion sicher vor einem Klartextangriff?

   (Lösung zu Kontrollfrage 6)

7. Berechnen Sie alle Werte von $8^a \pmod{11}$ für $a \in \mathbb{Z}_{11}^*$.

   (Lösung zu Kontrollfrage 7)

8. Was ist das Elgamal-Verfahren?

   (Lösung zu Kontrollfrage 8)

9. Warum muss Bob beim Elgamal-Verfahren für jede neue Nachricht an Alice eine neue Zufallszahl b für die Erzeugung des geheimen Schlüssels k verwenden?

   (Lösung zu Kontrollfrage 9)

10. Warum ist das Elgamal-Verfahren nicht für eine digitale Signatur geeignet?

    (Lösung zu Kontrollfrage 10)

11. Warum ist die diskrete Exponentialfunktion keine Einweg-**Falltür**funktion?

    (Lösung zu Kontrollfrage 11)

Fragen zu Abschnitt 5.2: Zyklische Gruppen

Erklären Sie folgende Begriffe und überprüfen Sie Ihre Antwort mit dem Skriptum: Ordnung, Generator, Square-and-Multiply, Zyklische Gruppe, Carmichael-Funktion.

1. Was ist die Ordnung eines Generators in $(\mathbb{Z}_p^*, \cdot)$?

   (Lösung zu Kontrollfrage 1)

2. Ist $(\mathbb{Z}_m, +)$ zyklisch? Wenn ja, was ist ein Generator?

   (Lösung zu Kontrollfrage 2)

3. Ist die Euler'sche φ-Funktion $\varphi(n)$ die kleinste Zahl x, die $a^x = 1 \pmod m$ für alle $a \in \mathbb{Z}_m$ erfüllt?

   (Lösung zu Kontrollfrage 3)

4. Was ist $\lambda(p_1 p_2)$ wenn $p_j = 2q_j + 1$ Safe Primes sind (also q_j wieder prim)?

   (Lösung zu Kontrollfrage 4)

5. Es seien p, q zwei Primzahlen. Wann ist $\mathbb{Z}_{pq}$ zyklisch?

   (Lösung zu Kontrollfrage 5)

6. Gibt es für $a \in \mathbb{Z}_m \setminus \mathbb{Z}_m^*$ ein (kleinstes) $j \in \mathbb{N}$ mit $a^j = 1 \pmod m$?

   (Lösung zu Kontrollfrage 6)

Fragen zu Abschnitt 5.3: Sicherheit des DH-Schlüsseltausch und des DLP

Erklären Sie folgende Begriffe und überprüfen Sie Ihre Antwort mit dem Skriptum: DLP

1. Was ist eine Sophie-Germain-Primzahl?

   (Lösung zu Kontrollfrage 1)

2. In welcher Gruppe funktioniert der Baby-Step-Giant-Step Algorithmus zur Lösung des DLP.

   (Lösung zu Kontrollfrage 2)

3. In welcher Gruppe funktioniert der Index Calculus zur Lösung des DLP.

   (Lösung zu Kontrollfrage 3)

Fragen zu Abschnitt 5.4: Der RSA-Verschlüsselungsalgorithmus

Erklären Sie folgende Begriffe und überprüfen Sie Ihre Antwort mit dem Skriptum: RSA-Algorithmus, Square-and-Multiply, Hybridverfahren.

1. Warum muss der öffentliche Schlüssel e beim RSA-Verfahren teilerfremd zu m gewählt werden?

   (Lösung zu Kontrollfrage 1)

2. Berechnen Sie $5^{16} \pmod{11}$ mit der Hand mithilfe des Square-and-Multiply Verfahrens.

   (Lösung zu Kontrollfrage 2)

3. Was ist ein Hybridverfahren?

   (Lösung zu Kontrollfrage 3)

4. Kann der RSA-Algorithmus zum digitalen Signieren von Dokumenten verwendet werden?

   (Lösung zu Kontrollfrage 4)

5. Was passiert bei RSA, wenn n das Produkt mehrere Primzahlen ist? Welche Vorteile/Nachteile hätte das?

   (Lösung zu Kontrollfrage 5)

6. Was passiert bei RSA, wenn $n = p$ eine Primzahl ist?

   (Lösung zu Kontrollfrage 6)

Fragen zu Abschnitt 5.5: Sicherheit des RSA-Algorithmus

Erklären Sie folgende Begriffe und überprüfen Sie Ihre Antwort mit dem Skriptum: Schlüssellänge, Faktorisierung, Zahlkörpersieb.

1. Worauf beruht die Sicherheit des RSA-Algorithmus?

   (Lösung zu Kontrollfrage 1)

2. Wenn im Jahr 2018 publiziert wird, dass die Faktorisierung einer 2048-Bit langen Primzahl mit 6-monatigem Aufwand auf 100 000 Workstations gelungen ist, soll man dann sofort alle 2048-Bit RSA-Schlüssel durch längere Schlüssel ersetzen?

   (Lösung zu Kontrollfrage 2)

3. Darf man bei RSA eine gemeinsame Zahl n erzeugen und dann mehreren Personen ihre Schlüsselpaare (n, d_j), (n, e_j) zuweisen? (Wenn p und q nach der Erzeugung der Schlüssel vernichtet werden.)

   (Lösung zu Kontrollfrage 3)

4. Erhöht bei RSA eine Mehrfachverschlüsselung mit verschiedenen e die Sicherheit?

   (Lösung zu Kontrollfrage 4)

Fragen zu Abschnitt 5.6: Primzahltests

Erklären Sie folgende Begriffe und überprüfen Sie Ihre Antwort mit dem Skriptum: Primzahltest, Test nach Miller und Rabin.

1. Wie viele Primzahlen gibt es ungefähr, die kleiner oder gleich als die Zahl 77732 sind?

   (Lösung zu Kontrollfrage 1)

2. Wie viele Primzahlen mit 3 Dezimalstellen gibt es näherungsweise (eine grobe Näherung genügt)?

   (Lösung zu Kontrollfrage 2)

3. Wie groß ist die Wahrscheinlichkeit (näherungsweise), durch zufällige Wahl einer 3-stelligen Dezimalzahl eine Primzahl zu treffen?

   (Lösung zu Kontrollfrage 3)

4. Wie wählt man in der Praxis die Primfaktoren für den RSA-Modul?

   (Lösung zu Kontrollfrage 4)

Lösungen zu den Kontrollfragen

Lösungen zu Abschnitt 5.1

1. Symmtrisches Verfahren: es gibt einen einzigen Schlüssel, der geheim ist, und der sowohl dem Sender als auch dem Empfänger bekannt sein muss. Mithilfe dieses Schlüssels wird verschlüsselt und entschlüsselt. Asymmetrisches Verfahren (auch Public-Key-Verfahren genannt): es gibt pro Teilnehmer zwei Schlüssel: einen öffentlichen Schlüssel (public key) und einen privaten Schlüssel (private key). Den öffentlichen Schlüssel kann jedermann verwenden, den privaten Schlüssel kennt nur sein Besitzer. Die Ermittlung des privaten Schlüssels aus dem öffentlichen Schlüssel ist innerhalb praktisch sinnvoller Zeitgrenzen nicht möglich.

2. Alice gibt ihren öffentlichen Schlüssel e bekannt, um Nachrichten zu empfangen. Wenn Bob ihr die Nachricht x verschlüsselt schicken möchte, so verwendet er Alices öffentlichen Schlüssel und berechnet $E_e(x)$. Alice entschlüsselt mithilfe ihres privaten Schlüssels: $x = D_d(E_e(x))$.

3. Eine Einweg-Falltürfunktion ist eine bijektive Funktion, die leicht auszuführen, aber schwer (praktisch nicht) invertierbar ist. Es gibt aber eine Geheiminformation, mit deren Hilfe man die Funktion leicht invertieren kann.

4. a) Man kann eine geheime Nachricht austauschen, ohne vorher einen geheimen Schlüssel vereinbaren zu müssen.
   b) Truhe, die während der Übertragung über den unsicheren Kanal immer durch zumindest ein Schloss gesichert ist.
   c) Eine Funktion, die (1) invertierbar, (2) bzgl. des Schlüssels kommutativ

und (3) sicher gegen einen Klartextangriff ist.
d) Es ist aufwendig, daher nicht gut zum Austausch langer Nachrichten geeignet, und es eignet sich nicht für spontanen Nachrichtenaustausch.

5. Das One-Time-Pad ist zwar kommutativ, aber nicht sicher gegen einen Klartextangriff. DES wäre sicher gegen einen Klartextangriff, ist aber nicht kommutativ.

6. a) Es ist eine Modifikation des No-Key-Protokoll: bei Diffie–Hellman wird aber nicht eine geheime Nachricht ver- und entschlüsselt, sondern nur ein gemeinsamer geheimer Schlüssel erzeugt, der dann für die Verschlüsselung der eigentlichen Nachricht weiterverwendet wird.
b) Die verwendete Funktion muss kommutativ und sicher gegen einen Klartextangriff sein.
c) Weil die diskrete Exponentialfunktion leicht auszuführen ist, ihre Umkehrfunktion, der diskrete Logarithmus, aber praktisch nicht. Zum Beispiel kann $\alpha = 3^6 \pmod{11}$ leicht berechnet werden (von Alice), aber es ist (für Eve) ungleich viel schwerer, aus $4 = 3^a \pmod{11}$ auf a zu kommen.

7. a durchläuft alle Zahlen aus $\mathbb{Z}_{11}^*$:

a	1	2	3	4	5	6	7	8	9	10
$y = 8^a \pmod{11}$	8	9	6	4	10	3	2	5	7	1

8. Elgamal ist ein Public-Key-Verfahren, bei dem ein gemeinsames Geheimnis (= Schlüssel k) mithilfe der diskreten Exponentialfunktion erzeugt wird. Der geheime Schlüssel k wird verwendet, um die eigentliche Nachricht mit einer Multiplikationschiffre zu verschlüsseln.

9. Wenn Eve mithilfe eines Klartextangriffes auf einen geheimen Schlüssel kommt, dann bleiben die Nachrichten, die mit einer anderen Zufallszahl verschlüsselt sind, sicher.

10. Wenn Alice ein Dokument mit dem geheimen Schlüssel k signiert, so kann nur Bob, der k gemeinsam mit Alice erzeugt hat, die Signatur prüfen, aber kein Dritter (was aber von einer digitalen Signatur, wie von jeder Unterschrift, verlangt wird).

11. Weil es keine Geheiminformation gibt, mit deren Hilfe ihre Umkehrung (= der diskrete Logarithmus) leicht berechnet werden kann.

Lösungen zu Abschnitt 5.2

1. $\varphi(p) = p - 1$.

2. Ja, ein Generator ist $a = 1$.

3. Nein, die kleinste Zahl ist gegeben durch die Carmichael-Funktion $\lambda(n)$.

4. $\lambda(p_1 p_2) = \mathrm{kgV}(2q_1, 2q_2) = 2q_1 q_2$.

5. $\mathbb{Z}_{pq}$ ist genau dann zyklisch wenn $\lambda(pq) = \mathrm{kgV}(p-1, q-1) = (p-1)(q-1) = \varphi(pq)$ gilt, wenn also $\mathrm{ggT}(p-1, q-1) = 1$ ist. Sind p und q ungerade, so ist immer 2 ein gemeinsamer Teiler von $p-1$ und $q-1$. $\mathbb{Z}_{pq}$ ist also genau dann zyklisch wenn $p = 2$ oder $q = 2$.

6. Nein: In diesem Fall würde ja $a = t\tilde{a}$ und $m = t\tilde{m}$ mit $t = \mathrm{ggT}(a, m) > 1$ und $\mathrm{ggT}(\tilde{a}, \tilde{m}) = 1$ gelten. Wir bräuchten also $a^j = t^j \tilde{a}^j = 1 + kt\tilde{m}$ und das ist ein Widerspruch, da t kein Teiler von 1 ist.

Lösungen zu Abschnitt 5.3

1. Eine Primzal p, für die auch $\frac{p-1}{2}$ prim ist.

2. In einer beliebigen zyklischen Gruppe.

3. Nur in $\mathbb{Z}_p^*$.

Lösungen zu Abschnitt 5.4

1. Weil es sonst keinen geheimen Schlüssel d dazu gibt. Denn $d = e^{-1} \pmod m$ existiert ja nur, wenn e und m teilerfremd sind.

2. $5^{16} = 5^{2\cdot2\cdot2\cdot2} = (5^2)^8 = (3^2)^4 = (9^2)^2 = 4^2 = 5 \pmod{11}$.

3. Ein Verfahren, bei dem RSA zuerst verwendet wird, um einen geheimen Schlüssel auszutauschen, und danach ein symmetrisches Verfahren (mit diesem geheimen Schlüssel), um die eigentliche Nachricht zu übermitteln.

4. Ja.

5. Solange man $m = \lambda(n)$ (bzw. $m = \varphi(n)$) verwendet, funktioniert alles weiterhin. Der Nachteil ist, dass das Faktorisieren von n durch mehrere Faktoren erleichtert wird. Der Vorteil, dass die Schlüsselerzeugung und das Entschlüsseln mit dem Chinesischen Restsatz bei kleineren Faktoren schneller geht.

6. Solange man $m = p - 1$ verwendet, funktioniert alles weiterhin, allerdings muss zum Ver-/Entschlüsseln p bekannt sein und es handelt sich daher um einen symmetrischen Algorithmus (der als **Pohlig-Hellmann-Verfahren**) bekannt ist.

Lösungen zu Abschnitt 5.5

1. Darauf, dass die Zerlegung einer hinreichend großen Zahl in ihre Primfaktoren auch mit den schnellsten verfügbaren Computern alle praktischen Zeitgrenzen übersteigt. Die Zerlegung des Moduls n in seinen Primfaktoren (und die anschließende Berechung des privaten Schlüssels) ist der einzige heute bekannte Angriff auf RSA.

2. Das hängt vom Wert der Daten ab und vom Zeitraum, für die die Daten geschützt werden müssen. Wenn die geheimen Daten Sicherheit für deutlich weniger als 6 Monate brauchen und ihr Wert deutlich weniger als die Kosten für das Laufen von 100.000 Workstations beträgt, so kann der 2048-Bit Schlüssel weiterhin genutzt werden. Für Daten, die länger gesichert sein sollen bzw. die mehr wert sind, sollte man zu einem längeren Schlüssel wechseln.

3. Nein, denn jeder Teilnehmer kann mit (n, d_j) und (n, e_j) die Primfaktoren bestimmen und damit die geheimen Schlüssel der anderen Teilnehmer berechnen.

4. Nein, da die Menge der RSA-Verschlüsselungsfunktionen eine Gruppe bilden: $E_e \circ E_{e'} = E_{ee'}$.

Lösungen zu Abschnitt 5.6

1. Ungefähr $\frac{77732}{\ln(77732)} = 6903$ (folgt aus dem Primzahlsatz).

2. Die Anzahl aller Primzahlen größer 99 und kleiner oder gleich 999 ist näherungsweise $\pi(999) - \pi(99) \approx \frac{999}{\ln(999)} - \frac{99}{\ln(99)} = 123$ (folgt wieder aus dem Primzahlsatz).

3. Die Wahrscheinlichkeit, unter den 900 dreistelligen Zahlen eine Primzahl zu treffen, ist näherungsweise gleich $\frac{123}{900} = 13.7\%$

4. Man wählt zufällig Zahlen in der gewünschten Größenordnung, solange, bis man auf eine Primzahl trifft. Der Test, ob es sich bei einer Zahl um eine Primzahl handelt, ist in der Regel ein statistischer Test.

5.8 Übungen

Aufwärmübungen

1. a) Wie erzeugt ein Element der Ordnung k eine Untergruppe der Ordnung k? Warum ist diese Untergruppe eine zyklische Gruppe?
   b) Was wissen wir über die möglichen Ordnungen von Elementen in einer Gruppe, wenn wir die Gruppenordnung kennen?
   c) Welche Ordnungen von Elementen sind daher möglich in der Gruppe
   (i) $\mathbb{Z}_5^*$ (ii) $\mathbb{Z}_7^*$ (iii) $\mathbb{Z}_{10}^*$ (iv) $\mathbb{Z}_{12}^*$?

2. a) Welche zwei Lösungskonzepte, die bis heute große praktische Bedeutung haben, haben Diffie und Hellman im Jahr 1976 vorgeschlagen, um das Problem zu lösen, dass bei einer symmetrischen Verschlüsselung der geheime Schlüssel in der Regel über einen unsicheren Kanal übertragen werden muss?
   b) Public Key Kryptosystem (PKC):
   (i) Wie viele Schlüssel besitzt jeder Teilnehmer? Wie heißen die Schlüssel und warum heißen sie so?
   (ii) Die Verschlüsselungsfunktion bei einem PKC ist eine Einweg-Falltürfunktion: Was bedeutet „Einweg", und was bedeutet „Falltür"? Wie lautet die englische Bezeichnung für eine Einweg-Falltürfunktion?

3. Diffie Hellman Key Exchange:
   a) Erklären (Skizzieren) Sie, wie das Protokoll mit einer allgemeinen Einwegfunktion $E_a(g)$ abläuft.
   b) Wird beim Protokoll die Umkehrfunktion der Einwegfunktion gebraucht?
   c) Welche Einwegfunktion wird konkret verwendet? Welche Richtung ist leicht, welche schwer?
   d) Ist auch die reelle (stetige) Exponentialfunktion eine Einwegfunktion? Warum?
   e) Warum ist es für die Sicherheit wesentlich, dass die Basis g der diskreten Exponentialfunktion möglichst viele Zahlen (d.h. eine möglichst große Untergruppe) erzeugt?

4. Für die Diffie–Hellmann Schlüsselvereinbarung sei $g = 6$ und $p = 11$:
   Alice wählt als geheime Zahl $a = 2$, Bob wählt $b = 7$. Geben Sie alle Schritte bzw. Zwischenergebnisse von Alice bzw. Bob an. Wie lautet der vereinbarte geheime Schlüssel?

5. Für die Diffie–Hellmann Schlüsselvereinbarung sei $g = 8$ und $p = 11$:
   Homer und Bart erzeugen einen gemeinsamen geheimen Schlüssel. Lisa belauscht die Kommunikation und erfährt dadurch $\alpha = 10$ und $\beta = 7$. Wie kommt sie (durch einen Brute Force Angriff) an den vereinbarten geheimen Schlüssel von Homer und Bart und wie lautet dieser geheime Schlüssel?

6. Für die Elgamal-Verschlüsselung sei $g = 8$, $p = 11$. Alice gibt als öffentlichen Schlüssel $\alpha = 7$ bekannt. Bob möchte die Nachricht $x = 5$ verschlüsselt an

Alice schicken. Wie geht er vor? (Gehen Sie davon aus, dass er als geheimen Schlüssel $b = 6$ wählt.)

7. Für die Elgamal-Verschlüsselung sei $g = 8$, $p = 11$. Alice gibt als öffentlichen Schlüssel $\alpha = 7$ bekannt. Sie erhält die verschlüsselte Nachricht $(c, \beta) = (9, 3)$ von Bob. Wie entschlüsselt sie (ihr geheimer Schlüssel ist $a = 9$)?

8. Für die Elgamal-Verschlüsselung sei $g = 6$, $p = 11$: Bart gibt als öffentlichen Schlüssel $\alpha = 9$ bekannt. Homer schickt an Bart $(\beta, c_1) = (7, 4)$ und $(\beta, c_2) = (7, 2)$ (er hat also offensichtlich den Fehler begangen und für beide Nachrichten dieselbe Zufallszahl verwendet). Lisa errät nun irgendwie, dass $c_1 = 4$ den Klartext $x_1 = 5$ bedeutet. Wie kann sie mit einem Klartextangriff den Schlüssel k ermitteln und somit auch c_2 entschlüsseln?

9. a) Wie erzeugt sich Alice zu $p = 11$ und $q = 17$ und zu $e = 3$ ihren zugehörigen privaten RSA-Schlüssel? (Warum wäre z.B. $e = 5$ kein zulässiger öffentlicher Schlüssel?)

   b) Bob möchte an Alice die Nachricht $x = 14$ verschlüsselt schicken. Wie geht er vor?
   c) Wie entschlüsselt Alice diese Nachricht? (Verwenden Sie z.B. Wolfram Alpha.)

10. Ihr geheimer RSA-Schlüssel sei $(n, d) = (209, 17)$, ihr öffentlicher Schlüssel ist $(n, e) = (209, 53)$. Verschlüsseln Sie die folgende Texte buchstabenweise und entschlüsseln Sie sie danach wieder (wandeln Sie zuvor die Nachricht in Zahlen $\{0, 1, 2, \ldots, 25\}$ um):
    a) „LINUX“, b) „ABBA“

Weiterführende Aufgaben

1. a) Geben Sie für jedes Element aus der additiven Gruppe $(\mathbb{Z}_5, +)$ dessen Ordnung an und die von diesem Element erzeugte Untergruppe.
   b) Warum ist $(\mathbb{Z}_m, +)$ für *beliebiges* $m \in \mathbb{N}$ zyklisch?

2. a) Bestimmen Sie für jedes $a \in \mathbb{Z}_{11}^*$ dessen Ordnung und die zyklische Untergruppe, die von diesem Element erzeugt wird. Machen Sie dazu eine Tabelle.
   b) Geben Sie alle Generatoren von $\mathbb{Z}_{11}^*$ an.
   c) Wie hängen die Ordnung eines Elementes und die Gruppenordnung zusammen?
   d) Wie kann man die Anzahl der Generatoren in einer zyklischen Gruppe berechnen?
   e) Wo sehen wir in der Tabelle, was der kleine Satz von Fermat allgemein aussagt?
   f) Gibt es zu jedem Teiler t von $|G|$ eine Untergruppe mit t Elementen? Begründen Sie, ob das allgemein der Fall ist.

3. a) Geben Sie für jedes Element aus $\mathbb{Z}_{12}^*$ seine Ordnung und die von diesem Element erzeugte Untergruppe an (machen Sie dazu eine Tabelle).
   b) Ist $\mathbb{Z}_{12}^*$ zyklisch?
   c) Überzeugen Sie sich anhand dieses Beispiels davon, dass Satz 5.10 (bzw. Satz 1.33) gilt.

4. a) Begründen Sie (ohne Rechnung), warum $\mathbb{Z}_{13}^*$ zyklisch ist.
   b) Wie viele Generatoren hat $\mathbb{Z}_{13}^*$?
   c) Geben Sie einen Generator von $\mathbb{Z}_{13}^*$ an. (Erstellen Sie dazu eine Tabelle solange, bis ein Generator gefunden ist.)
   d) Wie viele Untergruppen hat $\mathbb{Z}_{13}^*$?
   e) Geben Sie einen Generator der Untergruppe an, die 6 Elemente hat. (Tipp: Satz 5.20). Geben Sie weiters die Untergruppe mit 6 Elementen an.
   f) Wie viele Generatoren hat die Untergruppe mit 6 Elementen?

5. a) Zeigen Sie, dass in $\mathbb{Z}_m^*$ ($m > 2$) die Ordnung des Elementes $a = m - 1$ immer gleich 2 ist.
   b) Welche Untergruppe wird durch $a = m - 1$ erzeugt?

6. Wir betrachten die Gruppe $\mathbb{Z}_{23}^*$.
   a) Welche Ordnung hat $a = 2$? (Machen Sie dazu eine Tabelle für $a = 2$.)
   b) Geben Sie alle Untergruppen von $\mathbb{Z}_{23}^*$ an.
   c) Warum ist jedes Element der Untergruppe mit 11 Elementen (außer dem Element 1) Generator dieser Untergruppe?
   fd Geben Sie einen Generator von $\mathbb{Z}_{23}^*$ an.

7. Für die Diffie–Hellmann Schlüsselvereinbarung sei $g = 7$ und $p = 11$:
   Alice wählt als geheime Zahl $a = 3$, Bob wählt $b = 6$. Geben Sie alle Schritte bzw. Zwischenergebnisse von Alice bzw. Bob an. Wie lautet der vereinbarte geheime Schlüssel k?

8. Beweisen Sie, dass für $a, b, c \in G$ die Gleichung

   $$\log_a(b) \log_b(c) = \log_a(c) \pmod{\operatorname{ord}(a)}$$

   gilt, sofern beide Logarithmen existieren. Insbesondere gilt

   $$\log_a(b) \log_b(a) = 1 \pmod{\operatorname{ord}(a)}$$

   und in diesem Fall muss $\operatorname{ord}(a) = \operatorname{ord}(b)$ gelten.

9. Beweisen Sie, dass die Schwierigkeit des DLP in $\mathbb{Z}_p^*$ unabhängig vom verwendeten Generator ist.

10. Für die Diffie–Hellmann Schlüsselvereinbarung sei $g = 8$ und $p = 11$:
    Alice wählt als geheime Zahl $a = 3$, Bob wählt $b = 6$. Geben Sie alle Schritte bzw. Zwischenergebnisse von Alice bzw. Bob an. Wie lautet der vereinbarte geheime Schlüssel?

11. Mittels des Algorithmus **Square and Multiply** kann man Potenzen x^k effizient berechnen. Beispiel:

$$x^{23} = (((x^2)^2 \cdot x)^2 \cdot x)^2 \cdot x$$

Recherchieren Sie, wie der Algorithmus anzuwenden ist und berechnen Sie mit seiner Hilfe auf effiziente Weise:
a) x^{17} b) x^{107}

12. Homer hat sich eine Safe Prime $p = 1283$ und einen Generator $g = 2$ ausgedacht. Als privaten Schlüssel hat er $a = 641$ gewählt. Was halten Sie davon?

13. Für die Diffie–Hellmann Schlüsselvereinbarung sei $g = 4$ (also kein Generator in $\mathbb{Z}_{11}^*$) und $p = 11$: Homer und Bart erzeugen einen gemeinsamen Schlüssel, Lisa belauscht die Kommunikation und erfährt dadurch $\alpha = 9$ und $\beta = 5$. Wie kommt sie (durch einen Brute Force Angriff) nun besonders leicht an den geheimen Schlüssel von Homer und Bart und wie lautet dieser geheime Schlüssel?

14. Zeigen Sie: Jede Sophie-Germain-Primzahl $q > 3$ erfüllt $q = 5 \pmod 6$ und jede Safe-Prime $p > 7$ erfüllt $p = 11 \pmod{12}$.

15. Für die Elgamal-Verschlüsselung sei $g = 7$, $p = 11$. Alice gibt als öffentlichen Schlüssel $\alpha = 2$ bekannt.

a) Bob möchte die Nachricht $x = 8$ verschlüsselt an Alice schicken. Wie geht er vor? (Gehen Sie davon aus, dass er als geheimen Schlüssel $b = 6$ wählt.)

b) Alice erhält die verschlüsselte Nachricht $(c, \beta) = (6, 4)$ von Bob. Wie entschlüsselt sie (ihr geheimer Schlüssel ist $a = 3$)?

16. Für das Elgamal-Verfahren sei $g = 7$, $p = 11$: Bart gibt als öffentlichen Schlüssel $\alpha = 3$ bekannt. Homer schickt an Bart $(\beta, c_1) = (8, 2)$ und $(\beta, c_2) = (8, 9)$ (er hat also offensichtlich den Fehler begangen und für beide Nachrichten dieselbe Zufallszahl verwendet). Lisa erfährt nun irgendwie, dass $c_1 = 2$ den Klartext $x_1 = 6$ bedeutet. Wie kann sie mithilfe eines Klartextangriffes den Schlüssel k (oder gleich den inversen Schlüssel k^{-1}) ermitteln und somit auch c_2 entschlüsseln? Wie lautet x_2?

17. Zeigen Sie mithilfe des kleinen Satzes von Fermat, dass beim Verschlüsselungsverfahren von Elgamal der Schlüssel zum Entschlüsseln gleich

$$k^{-1} = \beta^{p-1-a} \pmod p$$

ist.

18. Berechnen Sie $\log_2(12)$ in $\mathbb{Z}_{13}^*$ mit der Pohlig-Hellman Reduktion.

19. Wie erzeugt Alice aus dem Primzahlpaar $p = 11$ und $q = 23$ einen öffentlichen und einen privaten RSA-Schlüssel?

20. a) Codieren Sie den Klartext

    ITS ALL GREEK TO ME

    gemäß:

    $$\text{Leerzeichen} = 00, \text{A} = 01, \text{B} = 02, \ldots, \text{Z} = 26$$

    Zerlegen Sie danach den Klartext in Blöcke der Länge 4.
    b) Verschlüsseln Sie den ersten Block mit RSA, indem Sie der öffentlichen Schlüssel $(n, e) = (2773, 17)$ verwenden.
    Tipp: https://people.csail.mit.edu/rivest/pubs/RSA78.pdf Seite 124

21. a) Wie erzeugt sich Alice zu $p = 11$ und $q = 19$ und zu $e = 7$ ihren zugehörigen privaten RSA-Schlüssel? (Warum wäre z.B. $e = 3$ kein zulässiger öffentlicher Schlüssel?)

    b) Bob möchte an Alice die Nachricht $x = 5$ verschlüsselt schicken. Wie geht er vor?
    c) Wie entschlüsselt Alice diese Nachricht? (Verwenden Sie z.B. Wolfram Alpha.)

22. Alice erhält von Bob die mit RSA verschlüsselte Nachricht $y = 43$.
    a) Entschlüsseln sie die Nachricht mit dem geheimen Schlüssel $(d, n) = (27, 55)$.
    b) Entschlüsseln Sie mit dem beschleunigten Verfahren indem Sie die Faktorisierung ($p = 5$, $q = 11$) und den Chinesischen Restsatz verwenden. Welche Teile können Sie vorberechnen und für zukünftige Entschlüsselungen speichern?
    c) Kann man diese Beschleunigung auch für das Verschlüsseln anwenden? Begründen Sie!

23. Alice besitzt den privaten RSA-Schlüssel $(n, d) = (1147, 149)$. Sie erhält von Bob den Geheimtext:

    233, 286, 815, 24, 187, 94, 992, 992, 187, 844, 919, 711, 103, 573, 418,
    286, 307, 187, 1067, 103, 591

    Dieser wurde mittels RSA und OAEP verschlüsselt. Dazu wurde die originale Nachricht zuerst in ASCII codiert (8 Bit Blöcke) und mittels OAEP zu 10 Bit Blöcken verschlüsselt. Die Umwandlung zwischen 10 Bit Blöcken und Zahlen erfolgt, indem man die 10 Bit als Stellen im Dualsystem betrachtet. Als Hashfunktion wurde $G(x) = (x^2 \bmod n) \bmod 2^8$ und $H(x) = (x^2 \bmod n) \bmod 2^2$ verwendet. (Hinweis: Falls Sie von Hand rechnen reicht es, den ersten Buchstaben zu entschlüsseln. Sie können aber auch ein kleines Programm schreiben.)

24. Homer und Bart haben die öffentlichen Schlüssel $(2021027, 17)$ und $(2021027, 23)$. Wie kann Bart aus seinem geheimen Schlüssel $(2021027, 135665)$ den von Homer berechnen (ohne Faktorisieren)?

25. Der öffentliche Schlüssel von Homer ist $(2029039, 1350683)$ und den privaten hat er verloren. Kann ihm (ohne Faktorisieren) geholfen werden? (Hinweis: Er kennt den Wiener-Angriff nicht.)

26. Angenommen Homer und Bart verwenden die öffentlichen Schlüssel (n, e_H) und (n, e_B) mit gleichem n. Marge sendet die gleiche Nachricht x an die beiden. Warum kann jeder aus $y_H = x^{e_H}$ und $y_B = x^{e_B}$ die Nachricht x berechnen, wenn e_H und e_B teilerfremd sind?

27. Der RAS-Modul sei $n = 14351 = 113 \cdot 127$ und der öffentliche Schlüssel $e = 5$.

    a) Was ist der Wiederherstellungsexponent von $x = 2$?
    b) Stellen Sie eine Liste aller möglichen Wiederherstellungsexponenten zusammen mit der Anzahl der Elemente die diese realisieren auf.

28. Beweisen Sie, dass RSA auch in dem Fall, dass n das Produkt mehrerer Primzahlen ist funktioniert. Darf eine Primzahl mehrfach vorkommen?

29. a) Es sei $n = p \cdot q$ das Produkt zweier Primzahlen $p > q$. Wie viele Möglichkeiten gibt es n in der Form $n = a^2 - b^2$ mit $a, b > 0$ zu schreiben?
    b) Faktorisieren Sie $n = 1363$ mit der Methode von Fermat.

30. Faktorisieren Sie $n = 38911$ mit der Methode von Fermat.

31. Faktorisieren Sie $n = 3473$ mit der $p - 1$ Methode.

32. Was passiert, wenn Alice bei der RSA-Schlüsselerzeugung irrtümlich nicht zwei Primzahlen verwendet? Betrachten wir ein einfaches Beispiel. Alice wählt $p = 12$, $q = 11$; dann ist $n = 132$ und $m = (p-1)(q-1) = 110$. Wenn sie als öffentlichen Schlüssel $e = 3$ wählt, ergibt sich für den geheimen Schlüssel $d = \frac{1}{3} \bmod 110 = 37$.
    a) Wie lautet der Geheimtext, wenn Bob den Klartext $INFORMATIK$, also (in ASCII) $73, 78, 70, 79, 82, 77, 65, 84, 73, 75$ mit Alices öffentlichem Schlüssel verschlüsselt?
    b) Wie lautet der „Klartext", wenn Alice diesen Geheimtext wieder entschlüsselt?

33. **Anzahl der Primzahlen mit Länge l Bit**: Zeigen Sie durch Verwendung der Näherung $\pi(n) \approx \frac{n}{\ln(n)}$, dass gilt:

$$\pi(2^l) - \pi(2^{l-1}) \approx \frac{2^l(l-2)}{l(l-1)\ln(4)}.$$

34. **Wahrscheinlichkeit, eine Primzahl der Länge l Bit zu ziehen**: Zeigen Sie durch Verwendung der Näherungsformel aus Aufgabe 33, dass die Wahrscheinlichkeit, bei zufälliger Wahl einer l-Bit Zahl eine Primzahl zu ziehen, näherungsweise durch folgende Formel angegeben werden kann:

$$\frac{\pi(2^l) - \pi(2^{l-1})}{2^l - 2^{l-1}} \approx \frac{l-2}{l \cdot (l-1) \cdot \ln(2)}.$$

ist.

35. a) Wie viele Primzahlen der Länge $l = 128$ Bit gibt es näherungsweise?
b) Wie groß ist die Wahrscheinlichkeit, bei zufälliger Wahl einer 128-Bit Zahl eine Primzahl zu treffen?

36. Testen Sie mithilfe des **Fermat-Test**, ob n eine Primzahl ist.
a) $n = 2821$ b) $n = 809$
Wählen Sie jeweils: $a = 2, 3, 5, 7$. Wie entscheidet der Test?

37. Testen Sie mithilfe des **Primzahltests von Miller–Rabin**, ob n eine Primzahl ist:
a) $n = 71$ b) $n = 91$

38. Recherchieren Sie folgende Begriffe:
a) Fermatsche Pseudoprimzahl zur Basis a.
b) Carmichael-Zahl
c) Mersenne-Primzahl
d) Sophie-Germain-Primzahl

Lösungen zu den Aufwärmübungen

1. –

2. a) Diffie Hellman Key Exchange; Public Key Kryptographie
b) (i) 2 Schlüssel; einen privaten (geheimen) und einen öffentlichen (wird bekannt gegeben)
(ii) Einweg: Eine Richtung ist leicht auszuführen, die andere schwer (es ist theoretisch möglich, aber praktisch zu aufwendig); Falltür: Zusatzinformation, mit deren Hilfe man die schwere Richtung der Einwegfunktion leicht durchführen kann; z.B. geheimer Schlüssel, mit dem man leicht entschlüsseln kann

3. b) Nein, niemand muss hier die Umkehrfunktion berechnen, daher braucht man für dieses Protokoll auch nur eine Einwegfunktion und nicht eine Einweg-*Falltürfunktion*.
c) Die diskrete Exponentialfunktion in der zyklischen Gruppe $\mathbb{Z}_p^*$. Leicht:

Potenzieren; schwer: den Exponenten (= Logarithmus) berechnen
d) Nein; denn aufgrund der Stetigkeit und strengen Monotonie kann man aus dem Funktionswert $y = g^x$ leicht auf $x = \log_g(y)$ rückschließen (d.h. den Logarithmus berechnen).
e) Weil dann ein Angreifer, der mittels Brute Force vorgeht, viele Möglichkeiten für den geheimen Exponenten durchprobieren muss.

4. Alice berechnet $\alpha = 6^2 = 3$ und schickt dies an Bob. Bob berechnet $\beta = 6^7 = 8$. Nun berechnen Alice und Bob dasselbe k: Alice berechnet $k = \beta^a = 8^2 = 9$ und Bob berechnet $\alpha^b = 3^7 = 9$.

5. Um die geheime Zahl a zu erhalten, muss sie die Gleichung $\alpha = g^a (\mathrm{mod}\, p)$ nach a auflösen (d.h. den diskreten Logarithmus von α ermitteln). Das ist hier durch Probieren möglich (Wertetabelle für $\alpha = 8^a (\mathrm{mod}\, 11)$ erstellen und das zu $\alpha = 10$ gehörige a ablesen). Es ergibt sich $a = 5$; analog $b = 9$; gemeinsames Geheimnis ist $k = \alpha^b = 10^9 = 10$. Probe: $k = \beta^a = 7^5 = 10$.

6. Bob wählt $b = 6$; es folgt $\beta = g^b = 8^6 = 3$, $k = \alpha^b = 7^6 = 4$, $c = kx = 4 \cdot 5 = 9$; Bob schickt $(c, \beta) = (9, 3)$ an Alice.

7. $k = \beta^a = 3^9 = 4$, $k^{-1} = 3$, $x = k^{-1}c = 3 \cdot 9 = 5$,

8. Lisa weiß, dass $4 = k \cdot 5$ und berechnet daraus $k = 5^{-1} \cdot 4 = 9 \cdot 4 = 3 (\mathrm{mod}\, 11)$. Sie berechnet weiters $k^{-1} = 4$, und damit $x_2 = k^{-1} c_2 = 8$.

9. a) $d = 107$
   b) $y = 126$
   c) $x = 126^{107} \bmod 187 = 14$

10. a) Wir wandeln zunächst „LINUX“ in Ziffern um: 11, 8, 13, 20, 23. Nun verschlüsseln wir $11^{53} (\mathrm{mod}\, 209) = 121, \ldots$ und erhalten als Geheimtext: 121, 50, 41, 58, 100.

    Zur Kontrolle entschlüsseln wir den Text wieder $121^{17} (\mathrm{mod}\, 209) = 11, \ldots$ und erhalten den Klartext.
    b) Wieder wandeln wir zunächst „ABBA“ in Ziffern um: 0, 1, 1, 0. Wegen $0^{53} (\mathrm{mod}\, 209) = 0$, $1^{53} (\mathrm{mod}\, 209) = 1$ ist der Geheimtext gleich dem Klartext.

    Da immer $0^e = 0 \,(\mathrm{mod}\, n)$ und $1^e = 1 \,(\mathrm{mod}\, n)$ gilt, werden die Zahlen 0 und 1 immer auf sich selbst abgebildet.

Lösungen zu ausgewählten Aufgaben

1. –

2. a)

a	ord(a)	von a erz. Untergr.
1	1	$\{1\}$
2	10	$\mathbb{Z}_{11}^*$
3	5	$\{1,3,4,5,9\}$
4	5	$\{1,3,4,5,9\}$
5	5	$\{1,3,4,5,9\}$
6	10	$\mathbb{Z}_{11}^*$
7	10	$\mathbb{Z}_{11}^*$
8	10	$\mathbb{Z}_{11}^*$
9	5	$\{1,3,4,5,9\}$
10	2	$\{1,10\}$

   b) Generatoren: $2, 6, 7, 8$
   c) Die Ordnung jedes Elementes ist ein Teiler der Gruppenordnung.
   d) In einer zyklischen Gruppe mit m Elementen gibt es genau $\varphi(m)$ Generatoren.
   e) In der Spalte von a^{10}.
   f) Es gibt zu jedem Teiler t von $|G|$ einer *zyklischen* Gruppe G genau eine Untergruppe mit t Elementen.

3. b) Da es keinen Generator gibt, ist $\mathbb{Z}_{12}^*$ nicht zyklisch.
   c) Die Gruppe hat 4 Elemente. In der Tabelle sehen wir, dass $a^4 = 1$ ist für jedes Element a der Gruppe.

4. a) Da 13 prim ist. b) 4. c) z.B. $g = 2$. d) 6. e) Generator 4, Untergruppe $4, 3, 12, 9, 10, 1$. f) 2.

5. a) — b) $\{1, m-1\}$

6. a) Ordnung 11
   b) $\{1\}$, $\{1, 22\}$, $\{1, 2, 3, 4, 6, 8, 9, 12, 13, 16, 18\}$, $\mathbb{Z}_{23}^*$
   c) Weil die Untergruppe prime Ordnung hat
   d) 5, 7, 10, 11, 14, 15, 17, 19, 20, 21 sind die Generatoren

7. $k = 9$

8. –

9. –

10. $k = 5$

11. a) $x^{17} = (((x^2)^2)^2)^2 \cdot x$
    b) $x^{107} = (((((x^2 \cdot x)^2)^2 \cdot x)^2)^2 \cdot x)^2 \cdot x$

12. Keine gute Wahl, da $\alpha = g^a$ Ordnung 2 hat.

13. Da g nur die Ordnung 5 hat, ist der Brute-Force Aufwand für Lisa geringer. Sie muss nur die von g erzeugte Untergruppe nach a bzw. b durchsuchen; $k = 4$.

14. –

15. a) Bob schickt $(c, \beta) = (6, 4)$ an Alice.
    b) Alice berechnet $k^{-1} = 5$ und $x = 8$.

16. $x_2 = 5$

17. Rechnen Sie nach, dass $k\beta^{p-1-a} = 1 \pmod{p}$ gilt.

18. 6

19. –

20. –

21. a) $d = 103$ b) $y = 168$ c) ...

22. a), b) $x = 32$ c) nein

23. It's all greek to me.

24. $1103015 = 94007 \pmod{\lambda(n)}$

25. $d = 3$

26. Hinweis: Erweiterter Euklid'scher Algorithmus mit e_H und e_B.

27. a) 6 b) 12 (z.B. $x = 3$).

28. Nein, eine Primzahl darf nicht mehrfach vorkommen.

29. a) Es gibt zwei Möglichkeiten. b) $n = 47 \cdot 29$.

30. $n = 233 \cdot 167$

31. $n = 23 \cdot 151$

32. Die Entschlüsselung liefert für einige Buchstaben nicht mehr den ursprünglichen Klartext.

33. Es sind einige elementare Umformungen (Bruchrechnen, Rechnen mit Logarithmen) notwendig:

$$\begin{aligned} \frac{2^l}{\ln(2^l)} - \frac{2^{l-1}}{\ln(2^{l-1})} &= \frac{2^l}{l \cdot \ln(2)} - \frac{2^{l-1}}{(l-1) \cdot \ln(2)} = \dots \\ &= \frac{2^l(l-2)}{l \cdot (l-1) \cdot \ln(4)}. \end{aligned}$$

34. Es sind nur elementare Umformungen (Bruchrechnen, Rechnen mit Logarithmen) notwendig.

35. a) $\frac{2^{128} \cdot 126}{128 \cdot 127 \ln(4)} = 1.9 \cdot 10^{36}$ b) $\frac{126}{128 \cdot 127 \cdot \ln(2)} = 0.0111823 = 1.1\%$

36. a) keine Primzahl
    b) Test entscheidet „wahrscheinlich Primzahl“

37. –

38. –

Literatur

[Adr+15] D. Adrian u. a. „Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice“. In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. CCS '15. New York, NY, USA: Association for Computing Machinery, 2015, S. 5–17. DOI: 10.1145/2810103.2813707. URL: https://dl.acm.org/doi/10.1145/2810103.2813707.

[Bon99] D. Boneh. „Twenty Years of Attacks on the RSA Cryptosystem“. In: *Notices of the AMS* 46 (1999), S. 203–213. URL: https://crypto.stanford.edu/~dabo/papers/RSA-survey.pdf.

[CP05] R. Crandall und C. Pomerance. *Prime Numbers, A Computational Perspective*. 2. Aufl. New York: Springer, 2005.

[DH76] W. Diffie und M. E. Hellman. „New directions in cryptography“. In: *IEEE Trans. Information Theory* IT-22.6 (1976), S. 644–654. URL: https://www-ee.stanford.edu/~hellman/publications/24.pdf.

[Eke21] M. Ekerå. „On completely factoring any integer efficiently in a single run of an order-finding algorithm“. In: *Quantum Information Processing* 20 (2021), S. 205. DOI: 10.1007/s11128-021-03069-1.

[FIPS186-5] NIST. „Digital Signature Standard (DSS)“. In: *Federal Information Processing Standards Publication (FIPS)* 186-5 (2023). URL: http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-5.pdf.

[For15] O. Forster. *Algorithmische Zahlentheorie*. 2. Aufl. Wiesbaden: Springer Spektrum, 2015.

[FPS01] J. B. Friedlander, C. Pomerance und I. E. Shparlinski. „Period of the power generator and small values of Carmichael's function“. In: *Math. Comput.* 70.236 (2001), S. 1591–1605. DOI: 10.1090/S0025-5718-00-01282-5.

[Fri+17] J. Fried u. a. „A Kilobit Hidden SNFS Discrete Logarithm Computation“. In: *Advances in Cryptology – EUROCRYPT 2017*. Hrsg. von J.-S. Coron und J. B. Nielsen. Cham: Springer, 2017, S. 202–231. DOI: 10.1007/978-3-319-56620-7_8.

[Gor84] J. Gorden. „Strong primes are easy to find“. In: *Workshop on the Theory and Application of of Cryptographic Techniques*. Bd. 209. Lecture Notes in Comput. Sci. Springer, Berlin, 1984, S. 216–223. DOI: 10.1007/3-540-39757-4_19.

[Gra08] A. Granville. „Smooth numbers, computational number theory and beyond“. In: *Algorithmic Number Theory: Lattices, Number Fields, Curves and Cryptography (MSRI Workshop)*. Bd. 44. Mathematical Sciences Research Institute Publications. 2008, S. 267–324. URL: http://library.msri.org/books/Book44/files/09andrew.pdf.

[Har+08] G. H. Hardy u. a. *An Introduction to the Theory of Numbers*. 6. Aufl. Oxford: Oxford UP, 2008.

[Hin09] M. J. Hinek. *Cryptoanalysis of RSA and its Variants*. CRC Press, 2009.

[Kat01] S. Katzenbeisser. *Recent Advances in RSA Cryptography*. Springer, 2001.

[Mau95] U. M. Maurer. „Fast generation of prime numbers and secure public-key cryptographic parameters“. In: *J. Cryptology* 8 (1995), S. 123–155. DOI: 10.1007/BF00202269.

[Mer78] R. C. Merkle. „Secure communications over insecure channels“. In: *Secure communications over insecure channels*. Bd. 21. Commun. of the ACM. 1978, S. 294–299. URL: http://www.merkle.com/1974/PuzzlesAsPublished.pdf.

[MOV96] A. J. Menezes, P. C. van Oorschot und S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[Nem+17] M. Nemec u. a. „The Return of Coppersmith's Attack: Practical Factorization of Widely Used RSA Moduli“. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS '17. Association for Computing Machinery, 2017, S. 1631–1648. DOI: 10.1145/3133956.3133969.

[RFC3526] T. Kivinen und M. Kojo. „More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)“. In: *Request for Comments* 3526 (2003). URL: https://www.rfc-editor.org/rfc/rfc3526 (besucht am 19. 02. 2018).

[RFC8017] K. M. Moriarty u. a. „PKCS #1: RSA Cryptography Specifications“. In: *Request for Comments* 8017 (2016). URL: https://www.rfc-editor.org/rfc/rfc8017.

[RS01] R. Rivest und R. Silverman. *Are 'Strong' Primes Needed for RSA*. Cryptology ePrint Archive, Paper 2001/007. 2001. URL: https://eprint.iacr.org/2001/007.

[Sho97a] P. W. Shor. „Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer“. In: *SIAM J. Comput.* 26.5 (1997), S. 1484–1509. DOI: 10.1137/S0097539795293172.

[Sho97b] V. Shoup. „Lower Bounds for Discrete Logarithms and Related Problems“. In: *Advances in Cryptology – EUROCRYPT '97*. Hrsg. von W. Fumy. Berlin: Springer, 1997, S. 256–266. DOI: 10.1007/3-540-69053-0_18.

[Wät08] D. Wätjen. *Kryptographie. Grundlagen, Algorithmen, Protokolle*. 2. Aufl. Spektrum, 2008.