Public-Key Verschlüsselung

8

8.1 Idee

Die Idee der Public-Key-Verschlüsselung wurde bereits in Abschn. 3.3 erwähnt. Dies wird wird hier noch einmal aufgegriffen und vertieft.

Ein zentrales Problem, das bei der Anwendung der bis jetzt beschriebenen symmetrischen Verschlüsselungsverfahren auftritt, ist die Verteilung und Verwaltung der Schlüssel. Immer wenn Alice und Bob miteinander geheim kommunizieren wollen, müssen sie vorher einen geheimen Schlüssel austauschen. Dafür muss ein sicherer Kanal zur Verfügung stehen. Ein Kurier muss den Schlüssel überbringen oder eine andere Lösung muss gefunden werden. Dieses Problem wird um so schwieriger, je mehr Teilnehmer in einem Netzwerk miteinander geheim kommunizieren wollen. Wenn es n Teilnehmer im Netz gibt und wenn alle miteinander vertraulich kommunizieren wollen, dann besteht eine Möglichkeit, das zu organisieren, darin, dass je zwei Teilnehmer miteinander einen geheimen Schlüssel austauschen. Dabei müssen dann $n(n-1)/2$ Schlüssel geheim übertragen werden, und genausoviele Schlüssel müssen irgendwo geschützt gespeichert werden. Im Januar 2015 gab es ungefähr drei Milliarden Internet-Nutzer. Wollten die alle einen Schlüssel austauschen wären das etwa $3 \cdot 10^{18}$ Schlüssel. Das wäre organisatorisch nicht zu bewältigen.

Eine andere Möglichkeit besteht darin, die gesamte Kommunikation über eine zentrale Stelle laufen zu lassen. Jeder Teilnehmer muss dann mit der Zentrale einen Schlüssel austauschen. Dieses Vorgehen setzt voraus, dass jeder Teilnehmer der Zentralstelle vertraut. Die Zentralstelle kennt ja alle geheimen Schlüssel und kann die gesamte Kommunikation mithören. Tatsächlich wird ein solcher Ansatz bei der Mobil-Telefonie (GSM – Global System for Mobile Communications) verwendet. Im GSM-System kommen viele verschiedene Zentralstellen zum Einsatz, die jeweils nur für einen beschränkten Kreis von Nutzerinnen und Nutzern zuständig sind und miteinander verbunden sind.

In diesem Kapitel präsentieren wir einen weiteren sehr wichtigen Ansatz: die Public-Key-Kryptographie. Die Idee dafür wurde 1976 von Diffie und Hellman [25] entwickelt. Kurz darauf schlugen Rivest, Shamir und Adleman [59] mit RSA ein solches Verfahren

J. Buchmann, *Einführung in die Kryptographie*, Springer-Lehrbuch,
DOI 10.1007/978-3-642-39775-2_8

Tab. 8.1 Verzeichnis öffentlicher Schlüssel

Name	öffentlicher Schlüssel
Buchmann	131213112359127531923751341 23
Bob	842283496450982361026311357 68
Alice	546282919826246381210250325 10
⋮	⋮

vor. Die Public-Key-Kryptographie vereinfacht das Schlüsselmanagment in großen Netzwerken erheblich und benötigt keine Zentralstellen, die alle Chiffretexte entschlüsseln können.

Die Idee der Public-Key-Kryptographie besteht darin, statt eines Schlüssels für Ver- und Entschlüsselung ein *Schlüsselpaar* (e, d) zu verwenden. Der Schlüssel e dient zum Verschlüsseln (englisch: Encryption) und kann öffentlich gemacht werden. Darum heißt er *öffentlicher Schlüssel*. Der Schlüssel d erlaubt es, die mit e erzeugten Chiffretexte wieder zu entschlüsseln (englisch: Decryption). Er muss genauso geheim bleiben, wie der Schlüssel eines symmetrischen Verschlüsselungsverfahrens. Darum heißt er *privater Schlüssel*. Der private Schlüssel darf also nicht mit vertrebarem Aufwand aus dem öffentlichen Schlüssel berechenbar sein.

Wie vereinfacht Public-Key-Kryptographie das Schlüsselmanagement? Wenn Alice vertrauliche Nachrichten erhalten möchte, erzeugt sie ein Schlüsselpaar (e, d) und veröffentlicht e in einem öffentlich zugänglichen Verzeichnis, wie es in Tab. 8.1 gezeigt ist. Bob verschlüsselt vertrauliche Nachrichten an Alice mit ihrem öffentlichem Schlüssel e. Alice verwendet ihren privaten Schlüssel d, um diese Nachrichten zu entschlüsseln. Der Austausch eines geheimen Schlüssels ist also nicht mehr erforderlich. Tatsächlich können alle, die vertrauliche Nachrichten für Alice verschlüsseln wollen, dazu denselben öffentlichen Schlüssel e verwenden. Da niemand außer Alice den privaten Schlüssel d kennt, kann nur sie alle diese Nachrichten lesen.

Bei Verwendung von Public-Key-Kryptographie wird der Austausch geheimer Schlüssel unnötig, aber die Authentizität der öffentlichen Schlüssel muss gewährleistet sein. Wer an Alice eine Nachricht schicken will, muss nämlich sicher sein, dass der öffentliche Schlüssel, den er aus dem öffentlichen Schlüsselverzeichnis erhält, tatsächlich der öffentliche Schlüssel von Alice ist. Mit diesem Thema beschäftigt sich Kap. 16. Gelingt es einem Angreifer, den öffentlichen Schlüssel von Alice in der Datenbank durch seinen eigenen zu ersetzen, dann kann er die Nachrichten, die für Alice bestimmt sind, lesen. Das öffentliche Schlüsselverzeichnis muss also vor Veränderung geschützt werden. Dazu werden elektronische Signaturen verwendet, die in Kap. 12 eingeführt werden.

Leider sind die bekannten Public-Key-Verfahren nicht so effizient wie viele symmetrische Verfahren. Darum benutzt man in der Praxis Kombinationen aus Public-Key-Verfahren und symmetrischen Verfahren. Eine Möglichkeit ist das sogenannte *Hybridverfahren*, das in Abschn. 3.3 beschrieben wurde.

8.2 Definition

Wir geben nun eine formale Definition von Public-Key-Verschlüsselungsverfahren. Sie ähnelt der Definition von Private-Key-Verschlüsselungsverfahren. Es gibt allerdings einige Unterschiede. Schlüssel werden durch Schlüsselpaare ersetzt. Sie bestehen aus einem öffentlichen Schlüssel und dem zugehörigen privaten Schlüssel. Bei Public-Key-Verfahren entspricht es der gängigen Praxis, Schlüssel in Abhängigkeit eines Sicherheitsparameters zu wählen. Die Schlüssel für die Public-Key-Verschlüsselungssysteme RSA und ElGamal können beliebig groß sein. Bei symmetrischen Verfahren ist das nicht der Fall. Die Schlüsselgröße von DES ist auf 56 Bit festgelegt. Die Schlüsselgröße von AES ist 128, 192 oder 256 Bit. Sobald diese Schlüsselgröße nicht mehr ausreicht, muss das AES-Verfahren modifiziert werden. Der Algorithmus zur Schlüsselerzeugung eines Public-Key-Verschlüsselungsverfahrens erhält also als Eingabe einen Sicherheitsparameter $k \in \mathbb{N}$. Er wählt die Größe der Schlüssel in Abhängigkeit von diesem Sicherheitsparameter. Tatsächlich ist die Eingabe der Bitstring 1^k, also die unäre Kodierung von k. Damit ist k die Eingabelänge des Algorithmus. Dies ist wichtig, weil verlangt wird, dass der Algorithmus polynomielle Laufzeit hat. Polynomiell bedeutet dann: polynomiell in k. Und das ist gewünscht.

Definition 8.1 Ein *Public-Key-Verschlüsselungsverfahren* oder *Public-Key-Kryptosystem* ist ein Tupel (**K**, **P**, **C**, **KeyGen**, **Enc**, **Dec**) mit folgenden Eigenschaften:

1. **K** ist eine Menge von Paaren. Sie heißt *Schlüsselraum*. Ihre Elemente (e, d) heißen *Schlüsselpaare*. Ist $(e, d) \in \mathbf{K}$, so heißt die Komponente e *öffentlicher Schlüssel* und die Komponente d *privater Schlüssel*.
2. **P** ist eine Menge. Sie heißt *Klartextraum*. Ihre Elemente heißen *Klartexte*.
3. **C** ist eine Menge. Sie heißt *Chiffretextraum*. Ihre Elemente heißen *Chiffretexte* oder *Schlüsseltexte*.
4. **KeyGen** ist ein probabilistischer Polynomzeit-Algorithmus. Er heißt *Schlüsselerzeugungsalgorithmus*. Bei Eingabe von 1^k für ein $k \in \mathbb{N}$ gibt er ein Schlüsselpaar $(e, d) \in K$ zurück. Wir schreiben dann $(e, d) \leftarrow \mathbf{KeyGen}(1^k)$. Mit $\mathbf{K}(k)$ bezeichnen wir die Menge aller Schlüsselpaare, die **KeyGen** bei Eingabe von 1^k zurückgeben kann. Mit $\mathbf{Pub}(k)$ bezeichnen wir die Menge aller öffentlichen Schlüssel, die als erste Komponente eines Schlüsselpaares $(e, d) \in \mathbf{K}(k)$ auftreten kann. Die Menge aller zweiten Komponenten dieser Schlüsselpaare bezeichnen wir mit $\mathbf{Priv}(k)$. Außerdem bezeichnet $\mathbf{P}(e) \subset \mathbf{P}$ die Menge der Klartexte, die mit einem öffentlichen Schlüssel e verschlüsselt werden können.
5. **Enc** ist ein probabilistischer Polynomzeit-Algorithmus. Er heißt *Verschlüsselungsalgorithmus*. Bei Eingabe von 1^k, $k \in \mathbb{N}$, eines öffentlichen Schlüssels $e \in \mathbf{Pub}(k)$ und eines Klartextes $P \in \mathbf{P}(e)$ gibt er einen Chiffretext C zurück. Wir schreiben dann $C \leftarrow \mathbf{Enc}(1^k, e, P)$.

6. **Dec** ist ein deterministischer Algorithmus. Er heißt *Entschlüsselungsalgorithmus*. Er entschlüsselt korrekt: Sei $k \in \mathbb{N}$, $(e,d) \in \mathbf{K}(k)$, $P \in \mathbf{P}(e)$ und $C \leftarrow \mathbf{Enc}(1^k, e, P)$. Dann gilt $P \leftarrow \mathbf{Dec}(1^k, d, C)$.

8.2.1 Sicherheit

In Abschn. 3.4 wurde die Sicherheit von symmetrischen Verschlüsselungsverfahren modelliert. Das Sicherheitsmodell für Public-Key-Verfahren ist dem ähnlich. Der entscheidende Unterschied besteht aber darin, dass Angreifer von Public-Key-Verfahren den öffentlichen Schlüssel kennen.

Genauso wie bei symmetrischen Verschlüsselungsverfahren ist es das Ziel eines Angreifers auf ein Public-Key-Kryptosystem aus Chiffretexten möglichst viel über die entsprechenden Klartexte zu erfahren. Dazu kann der Angreifer versuchen, private Schlüssel zu berechnen oder Informationen über die Klartexte zu gewinnen, die zu einzelnen Schlüsseltexten gehören.

Auch die Angriffsmethoden sind ähnlich wie bei symmetrischen Kryptosystemen. Allerdings müssen wir keine Ciphertext-Only-Angiffe berücksichtigen. Alle Angreifer kennen den öffentlichen Schlüssel. Sie können ihn also benutzen, um Klartexte ihrer Wahl zu verschlüsseln. Alle Angreifer können also mindestens Chosen-Plaintext-Angriffe ausführen. Auch Chosen-Ciphertext-Angriffe sind möglich. Allerdings müssen Angreifer dazu zeitweise die Möglichkeit haben, Chiffretexte zu entschlüsseln ohne den entsprechenden privaten Schlüssel zu kennen.

Bevor in Abschn. 8.5 Sicherheitsmodelle für Public-Key-Verschlüsselungsver fahren diskutiert werden, besprechen wir im nächsten Abschnitt zunächst das berühmteste Public-Key-Verfahren RSA.

8.3 Das RSA-Verfahren

Das RSA-Verfahren, benannt nach seinen Erfindern Ron Rivest, Adi Shamir und Len Adleman, war das erste Public-Key Verschlüsselungsverfahren und ist noch heute das wichtigste. Seine Sicherheit hängt eng mit der Schwierigkeit zusammen, große Zahlen in ihre Primfaktoren zu zerlegen.

8.3.1 Schlüsselerzeugung

Wir erklären, wie ein RSA-Schlüsselpaar erzeugt wird. Der Sicherheitsparameter sei eine Zahl $k \in \mathbb{N}$.

Der Schlüsselerzeugungsalgorithmus wählt zwei Primzahlen p und q (siehe Abschn. 7.5) mit der Eigenschaft, dass der *RSA-Modul*

$$n = pq$$

eine k-Bit-Zahl ist. Zusätzlich wählt der Algorithmus eine natürliche Zahl e mit

$$1 < e < \varphi(n) = (p-1)(q-1) \text{ und } \gcd(e,(p-1)(q-1)) = 1 \tag{8.1}$$

und berechnet eine natürliche Zahl d mit

$$1 < d < (p-1)(q-1) \text{ und } de \equiv 1 \bmod (p-1)(q-1). \tag{8.2}$$

Da $\gcd(e,(p-1)(q-1)) = 1$ ist, gibt es eine solche Zahl d tatsächlich. Sie kann mit dem erweiterten euklidischen Algorithmus berechnet werden (siehe Abschn. 1.6.3). Man beachte, dass e stets ungerade ist. Der öffentliche Schlüssel ist das Paar (n,e). Der private Schlüssel ist d.

Beispiel 8.1 Sei $k = 8$. Als Primzahlen wählt der Algorithmus die Zahlen $p = 11$ und $q = 23$. Also ist $n = 253 = 11111101$ ist, und $(p-1)(q-1) = 10 * 22 = 4 \cdot 5 \cdot 11$. Das kleinstmögliche e ist $e = 3$. Der erweiterte euklidische Algorithmus liefert $d = 147$.

Der RSA-Schlüsselraum besteht also aus allen Paaren $((n,e),d)$, die bei der obigen Konstruktion entstehen. Damit RSA sicher ist, müssen die Primfaktoren p und q geeignet gewählt werden. Üblich ist folgendes: der Sicherheitsparameter k wird als gerade Zahl gewählt, die heutzutage mindestens 1024 Bit lang ist. Die Primzahlen p und q werden wie in Abschn. 7.5 beschrieben als $k/2$-Bit-Zufallsprimzahlen gewählt. Dies wird in Abschn. 8.3.4 genauer diskutiert.

8.3.2 Verschlüsselung

Der RSA-Klartextraum ist $\mathbb{Z}_n$. Will Alice einen Klartext $m \in \mathbb{Z}_m$ für Bob verschlüsseln, besorgt sie sich seinen öffentlichen Schlüssel (e,n) und berechnet den Chiffretext

$$c = m^e \bmod n. \tag{8.3}$$

Alle, die den öffentlichen Schlüssel (n,e) kennen, können diese Verschlüsselung durchführen. Bei der Berechnung von $m^e \bmod n$ wird schnelle Exponentiation verwendet (siehe Abschn. 2.12), damit die Verschlüsselung schnell genug geht. Die Performanz von RSA wird in Abschn. 8.3.8 erläutert.

Beispiel 8.2 Wie in Beispiel 8.1 ist $n = 253$ und $e = 3$. Der Klartextraum ist also $\{0, 1, \ldots, 252\}$. Die Zahl $m = 165$ wird zu $165^3 \bmod 253 = 110$ verschlüsselt.

Das beschriebene RSA-Verfahren kann nur sehr kurze Klartexte verschlüsseln. Wir zeigen nun, wie man eine Art RSA-Blockchiffre realisieren kann, die zur Verschlüsselung beliebig langer Klartexte verwendet werden kann. Dies ist eher von theoretischem Interesse, weil die RSA-Verschlüsselung langer Klartexte viel langsamer ist als die Verschlüsselung mit symmetrischen Chiffren. In der Praxis wird statt dessen die in Abschn. 3.3 dargestellte Hybridverschlüsselung verwendet. Dabei wird ein symmetrischer Schlüssel RSA-verschlüsselt. Lange Klartexte werden mit diesem Schlüssel symmetrisch verschlüsselt. Der Empfänger entschlüsselt zuerst den symmetrischen Schlüssel und dann den symmetrisch verschlüsselten Klartext.

Wir nehmen an, dass das verwendete Alphabet Σ genau N Zeichen hat und die Zeichen die Zahlen $0, 1, \ldots, N-1$ sind. Wir setzen

$$r = \lfloor \log_N n \rfloor. \tag{8.4}$$

Ein Block $m_1 \ldots m_r$, $m_i \in \Sigma$, $1 \leq i \leq r$, wird in die Zahl

$$m = \sum_{i=1}^{r} m_i N^{r-i}$$

verwandelt. Man beachte, dass wegen (8.4)

$$0 \leq m \leq (N-1) \sum_{i=1}^{r} N^{r-i} = N^r - 1 < n$$

gilt. Im folgenden werden wir die Blöcke mit den durch sie dargestellten Zahlen identifizieren. Der Block m wird verschlüsselt, indem $c = m^e \bmod n$ bestimmt wird. Die Zahl c kann dann wieder zur Basis N geschrieben werden. Die N-adische Entwicklung von c kann aber die Länge $k+1$ haben. Wir schreiben also

$$c = \sum_{i=0}^{r} c_i N^{k-i} \quad c_i \in \Sigma, 0 \leq i \leq r.$$

Der Schlüsseltextblock ist dann

$$c = c_0 c_1 \ldots c_r.$$

In der beschriebenen Weise bildet RSA Blöcke der Länge r injektiv auf Blöcke der Länge $r+1$ ab. Dies ist keine Blockchiffre im Sinne von Definition 3.4. Trotzdem kann man mit der beschriebenen Blockversion des RSA-Verfahrens den ECB-Mode und den CBC-Mode (mit einer kleinen Modifikation) zum Verschlüsseln anwenden. Es ist aber nicht möglich, den CFB-Mode oder den OFB-Mode zu benutzen, weil in beiden Modes nur die Verschlüsselungsfunktion verwendet wird, und die ist ja öffentlich. Also kennt sie auch jeder Angreifer.

Beispiel 8.3 Wir setzen Beispiel 8.1 fort. Verwende $\Sigma = \{0, a, b, c\}$ mit der Entsprechung

0	a	b	c
0	1	2	3

Bei Verwendung von $n = 253$ ist $k = \lfloor \log_4 253 \rfloor = 3$. Das ist die Länge der Klartextblöcke. Die Länge der Schlüsseltextblöcke ist also 4. Es soll der Klartextblock abb verschlüsselt werden. Dieser entspricht dem String 122 und damit der Zahl

$$m = 1 * 4^2 + 2 * 4^1 + 2 * 4^0 = 26.$$

Diese Zahl wird zu

$$c = 26^3 \bmod 253 = 119$$

verschlüsselt. Schreibt man diese zur Basis 4, so erhält man

$$c = 1 * 4^3 + 3 * 4^2 + 1 * 4^1 + 3 * 4^0.$$

Der Schlüsseltextblock ist dann

$$acac.$$

Der hier beschriebene Verschlüsselungsalgorithmus ist deterministisch. Wie in Abschn. 3.4.2 erwähnt, kann diese Variante von RSA nicht sicher sein. Sichere Varianten werden in Abschn. 8.3.10 beschrieben.

8.3.3 Entschlüsselung

Die Entschlüsselung von RSA beruht auf folgendem Satz.

Theorem 8.1 *Sei (n, e) ein öffentlicher und d der entsprechende private Schlüssel im RSA-Verfahren. Dann gilt*

$$(m^e)^d \bmod n = m$$

für jede natürliche Zahl m mit $0 \leq m < n$.

Beweis Da $ed \equiv 1 \bmod (p-1)(q-1)$ ist, gibt es eine ganze Zahl l, so dass

$$ed = 1 + l(p-1)(q-1)$$

ist. Daher ist

$$(m^e)^d = m^{ed} = m^{1+l(p-1)(q-1)} = m(m^{(p-1)(q-1)})^l.$$

Aus dieser Gleichung erkennt man, dass

$$(m^e)^d \equiv m(m^{(p-1)})^{(q-1)l} \equiv m \bmod p$$

gilt. Falls p kein Teiler von m ist, folgt diese Kongruenz aus dem kleinen Satz von Fermat (siehe Theorem 2.13). Andernfalls ist die Behauptung trivial. Dann sind nämlich beide Seiten der Kongruenz 0 mod p. Genauso sieht man ein, dass

$$(m^e)^d \equiv m \bmod q$$

gilt. Weil p und q verschiedene Primzahlen sind, erhält man also

$$(m^e)^d \equiv m \bmod n.$$

Da $0 \leq m < n$ ist, erhält man die Behauptung des Satzes. □

Wurde also c wie in (8.3) berechnet, kann m mittels

$$m = c^d \bmod n$$

rekonstruiert werden. Damit ist gezeigt, dass das RSA-Verfahren tatsächlich ein Public-Key-Kryptosystem ist.

Beispiel 8.4 Wir schließen die Beispiele 8.1 und 8.3 ab. Dort wurde ja $n = 253$, $e = 3$ und $d = 147$ gewählt. Außerdem wurde $c = 119$ berechnet. Tatsächlich gilt $119^{147} \bmod 253 = 26$. Damit ist der Klartext rekonstruiert.

8.3.4 Sicherheit des privaten Schlüssels

Damit RSA sicher ist, muss es praktisch unmöglich sein, aus dem öffentlichen Schlüssel (n, e) den privaten Schlüssel d zu berechnen. In diesem Abschnitt werden wir zeigen, dass die Bestimmung des privaten Schlüssels d aus dem öffentlichen Schlüssel (n, e) genauso schwierig ist, wie die Zerlegung des RSA-Moduls n in seine Primfaktoren. Was ist die Bedeutung dieses Ergebnisses? Niemand kann beweisen, dass RSA sicher ist. Also möchte man möglichst viele und starke Argumente sammeln, die die Sicherheit von RSA belegen. Ein solches Argument lautet: Das Faktorisierungsproblem gilt seit Jahrhunderten als schwierig (siehe Kap. 9). Das Finden des privaten RSA-Schlüssels ist genauso schwierig. Ist das Argument überzeugend? Es ist ein Indiz für die Sicherheit von RSA, beweist sie aber nicht, denn es lässt Gegenargumente zu. Erstens kann RSA vielleicht gebrochen werden, ohne den privaten Schlüssel zu finden. In den folgenden Abschnitten werden wir zeigen, dass das unter bestimmten Bedingungen tatsächlich möglich ist. Zweitens ist nicht sicher, dass Faktorisieren schwer ist. Seit langem ist zum Beispiel bekannt, dass Quantencomputer das Faktorisierungsproblem in Polynomzeit lösen können (siehe [69]).

Trotzdem gilt: sind die RSA-Parameter richtig gewählt und wird eine sichere RSA-Variante verwendet, so besteht die einzige heute bekannte Möglichkeit, RSA auf einem klassischen Computer zu brechen, darin, den RSA-Modul zu faktorisieren.

Die beschriebene Eigenschaft teilt das RSA-Verfahren mit den anderen Public-Key-Verfahren. Ihre Sicherheit steht in engem Zusammenhang mit der Lösbarkeit schwieriger Berechnungsprobleme von allgemeinem mathematischen Interesse. Es ist aber nicht klar, dass diese Berechnungsprobleme wirklich schwierig sind.

Wir beweisen jetzt die angekündigte Äquivalenz.

Wenn der Angreifer Oskar die Primfaktoren p und q des RSA-Moduls n kennt, kann er aus n und dem Verschlüsselungsexponenten e den privaten Schlüssel d durch Lösen der Kongruenz $de \equiv 1 \bmod (p-1)(q-1)$ berechnen.

Wir zeigen, dass die Umkehrung auch gilt, wie man nämlich aus n, e, d die Faktoren p und q berechnet.

Dazu setzen wir

$$s = \max\{t \in \mathbb{N} : 2^t \text{ teilt } ed-1\}.$$

und

$$k = (ed-1)/2^s.$$

Lemma 8.1 *Für alle zu n teilerfremden ganzen Zahlen a gilt* $\text{order}(a^k + n\mathbb{Z}) \in \{2^i : 0 \leq i \leq s\}$.

Beweis Sei a eine zu n teilerfremde ganze Zahl. Nach Theorem 8.1 gilt $a^{ed-1} \equiv 1 \bmod n$. Daraus folgt $(a^k)^{2^s} \equiv 1 \bmod n$. Also impliziert Theorem 2.9, dass die Ordnung von $a^k + n\mathbb{Z}$ ein Teiler von 2^s ist. □

Der Algorithmus, der n faktorisiert, beruht auf folgendem Theorem.

Theorem 8.2 *Sei a eine zu n teilerfremde ganze Zahl. Wenn die Ordnung von* $a^k \bmod p$ *und* $\bmod q$ *verschieden ist, so ist* $1 < \gcd(a^{2^t k} - 1, n) < n$ *für ein* $t \in \{0, 1, 2, \ldots, s-1\}$.

Beweis Nach Lemma 8.1 liegt die Ordnung von $a^k \bmod p$ und $\bmod q$ in der Menge $\{2^i : 0 \leq i \leq s\}$. Sei die Ordnung von $a^k \bmod p$ größer als die von $a^k \bmod q$. Die Ordnung von $a^k \bmod q$ sei 2^t. Dann gilt $t < s$, $a^{2^t k} \equiv 1 \bmod q$, aber $a^{2^t k} \not\equiv 1 \bmod p$ und daher $\gcd(a^{2^t k} - 1, n) = q$. □

Um n zu faktorisieren, wählt man zufällig und gleichverteilt eine Zahl a in der Menge $\{1, \ldots, n-1\}$. Dann berechnet man $g = \gcd(a, n)$. Ist $g > 1$, so ist g ein echter Teiler von n. Der wurde ja gesucht. Also ist der Algorithmus fertig. Ist $g = 1$, so berechnet man

$$g = \gcd(a^{2^t k} - 1, n), \quad t = s-1, s-2, \ldots, 0.$$

Findet man dabei einen Teiler von n, dann ist der Algorithmus fertig. Andernfalls wird ein neues a gewählt und dieselben Operationen werden für dieses a ausgeführt. Wir wollen jetzt zeigen, dass in jeder Iteration dieses Verfahrens die Wahrscheinlichkeit dafür, dass ein Primteiler von n gefunden wird, wenigstens $1/2$ ist. Die Wahrscheinlichkeit dafür, dass das Verfahren nach r Iterationen einen Faktor gefunden hat, ist dann mindestens $1 - 1/2^r$.

Theorem 8.3 *Die Anzahl der zu n primen Zahlen a in der Menge* $\{1, 2, \ldots, n-1\}$, *für die* $a^k \bmod p$ *und* $\bmod q$ *eine verschiedene Ordnung hat, ist wenigstens* $(p-1)(q-1)/2$.

Beweis Sei g eine Primitivwurzel mod p und mod q. Eine solche existiert nach dem chinesischen Restsatz 2.18.

Zuerst nehmen wir an, die Ordnung von $g^k \bmod p$ sei größer als die Ordnung von $g^k \bmod q$. Diese beiden Ordnungen sind nach Lemma 8.1 Potenzen von 2. Sei x eine ungerade Zahl in $\{1, \ldots, p-1\}$ und sei $y \in \{0, 1, \ldots, q-2\}$. Sei a eine Lösung der simultanen Kongruenz

$$a \equiv g^x \bmod p, \quad a \equiv g^y \bmod q. \tag{8.5}$$

Dann ist die Ordnung von $a^k \bmod p$ dieselbe wie die Ordnung von $g^k \bmod p$. Die Ordnung von $a^k \bmod q$ ist aber höchstens so groß wie die Ordnung von $g^k \bmod q$, also kleiner als die Ordnung von $a^k \bmod p$. Schließlich sind diese Lösungen mod n paarweise verschieden, weil g eine Primitivwurzel mod p und q ist. Damit sind $(p-1)(q-1)/2$ zu n teilerfremde Zahlen a gefunden, für die die Ordnung von $a^k \bmod p$ und q verschieden ist.

Ist die Ordnung von $g^k \bmod q$ größer als die mod p, so geht man genauso vor.

Sei schließlich angenommen, dass die Ordnung von $g^k \bmod p$ und mod q gleich ist. Da $p-1$ und $q-1$ beide gerade sind, ist diese Ordnung wenigstens 2. Wir bestimmen die gesuchten Zahlen a wieder als Lösung der simultanen Kongruenz (8.5). Die Exponentenpaare (x, y) müssen diesmal aber aus einer geraden und einer ungeraden Zahl bestehen. Es bleibt dem Leser überlassen, zu verifizieren, dass es $(p-1)(q-1)/2 \bmod n$ verschiedene Lösungen gibt, und diese die gewünschte Eigenschaft haben. □

Aus Theorem 8.3 folgt unmittelbar, dass die Erfolgswahrscheinlichkeit des Faktorisierungsverfahrens in jeder Iteration wenigstens $1/2$ ist.

Beispiel 8.5 In Beispiel 8.1 ist $n = 253$, $e = 3$ und $d = 147$. Also ist $ed - 1 = 440$. Wenn man $a = 2$ verwendet, so erhält man $\gcd(2^{220}-1, 253) = \gcd(2^{110}-1, 253) = 253$. Aber $\gcd(2^{55} - 1, 253) = 23$.

8.3.5 Auswahl von p und q

Um die Faktorisierung des RSA-Moduls schwer genug zu machen, müssen RSA-Moduln hinreichend groß gewählt werden. Nach dem Moorschen Gesetz verdoppelt sich die Anzahl der Operationen, die Computer pro Zeiteinheit ausführen können, alle 18 Monate. Die Mindestgröße der RSA-Moduln muss also entsprechend vergrößert werden. Tab. 8.2 zeigt empfohlene Mindestgrößen Für RSA-Moduln. Die Werte wurden im Rahmen des EU-Projekts ECRYPT II ermittelt (siehe [73]).

Tab. 8.2 Mindestgröße von RSA-Moduln

Schutz bis	Mindestgröße
2015	1248
2020	1776
2030	2432
2040	3248
für absehbare Zukunft	15.424

Die Primfaktoren p und q werden solange als zufällige $k/2$-Bit-Primzahlen gewählt, bis $n = pq$ eine k-Bit-Zahl ist. Die zufällige Wahl von p und q stellt sicher, dass keine Faktorisierungsalgorithmen verwendet werden können, die die spezielle Struktur der Faktoren ausnutzen. Ein Beispiel für einen solchen Algorithmus ist das Pollard-$p-1$-Verfahren aus Abschn. 9.2.

8.3.6 Auswahl von e

Der öffentliche Schlüssel e wird so gewählt, dass die Verschlüsselung effizient möglich ist, ohne dass die Sicherheit gefährdet wird. Die Wahl von $e = 2$ ist natürlich immer ausgeschlossen, weil $\varphi(n) = (p-1)(q-1)$ gerade ist und $\gcd(e, (p-1)(q-1)) = 1$ gelten muss. Der kleinste mögliche Verschlüsselungsexponent ist also $e = 3$, wenn $\gcd(3, (p-1)(q-1)) = 1$ ist. Verwendet man diesen Exponenten, so benötigt die Verschlüsselung nur eine Quadrierung und eine Multiplikation mod n.

Beispiel 8.6 Sei $n = 253$, $e = 3$, $m = 165$. Um $m^e \bmod n$ zu berechnen, bestimmt man $m^2 \bmod n = 154$. Dann berechnet man $m^3 \bmod n = ((m^2 \bmod n) * m) \bmod n = 154 * 165 \bmod 253 = 110$.

Die Verwendung des Verschlüsselungsexponenten $e = 3$ ist aber nicht ungefährlich. Ein Angreifer kann nämlich der sogenannte *Low-Exponent-Angriff* anwenden. Diese beruht auf folgendem Satz.

Theorem 8.4 *Seien $e \in \mathbb{N}$, $n_1, n_2, \ldots, n_e \in \mathbb{N}$ paarweise teilerfremd und $m \in \mathbb{N}$ mit $0 \leq m < n_i$, $1 \leq i \leq e$. Sei $c \in \mathbb{N}$ mit $c \equiv m^e \bmod n_i$, $1 \leq i \leq e$, und $0 \leq c < \prod_{i=1}^{e} n_i$. Dann folgt $c = m^e$.*

Beweis Die Zahl $c' = m^e$ erfüllt die simultane Kongruenz $c' \equiv m^e \bmod n_i$, $1 \leq i \leq e$ und es gilt $0 \leq c' < \prod_{i=1}^{e} n_i$, weil $0 \leq m < n_i$, $1 \leq i \leq e$, vorausgesetzt ist. Da eine solche Lösung der simultanen Kongruenz aber nach dem chinesischen Restsatz eindeutig bestimmt ist, folgt $c = c'$. □

Der Low-Exponent-Angriff nutzt Theorem 8.4 aus. Er ist anwendbar, wenn die Voraussetzungen aus diesem Theorem erfüllt sind, wenn also ein Klartext m mit demselben öffentlichen Exponenten e und e zueinander paarweise teilerfremden Moduln verschlüs-

selt wird. Es ist zum Beispiel denkbar, dass eine Bank an e verschiedene Kunden dieselbe Nachricht verschlüsselt sendet. Dabei werden die verschiedenen öffentlichen Schlüssel n_i, $1 \leq i \leq e$, der Kunden benutzt, aber immer derselbe Verschlüsselungsexponent e. Dann kann der Angreifer den Klartext m folgendermaßen berechnen. Er kennt die Schlüsseltexte $c_i = m^e \bmod n_i$, $1 \leq i \leq e$. Mit dem chinesischen Restsatz berechnet er eine ganze Zahl c mit $c \equiv c_i \bmod n_i$, $1 \leq i \leq e$ und $0 \leq c < \prod_{i=1}^{e} n_i$. Nach Theorem 8.4 gilt $c = m^e$. Also kann der Angreifer m finden, indem er aus c die e-te Wurzel zieht. Dies ist in Polynomzeit möglich.

Beispiel 8.7 Wir wählen $e = 3$, $n_1 = 143$, $n_2 = 391$, $n_3 = 899$, $m = 135$. Dann ist $c_1 = 60$, $c_2 = 203$, $c_3 = 711$. Um den chinesischen Restsatz zu verwenden berechne x_1, x_2, x_3 mit $x_1 n_2 n_3 \equiv 1 \bmod n_1$, $n_1 x_2 n_3 \equiv 1 \bmod n_2$ und $n_1 n_2 x_3 \equiv 1 \bmod n_3$. Es ergibt sich $x_1 = -19$, $x_2 = -62$, $x_3 = 262$. Dann ist $c = (c_1 x_1 n_2 n_3 + c_2 n_1 x_2 n_3 + c_3 n_1 n_2 x_3) \bmod n_1 n_2 n_3 = 2460375$ und $m = 2460375^{1/3} = 135$.

Man kann den Low-Exponent-Angriff verhindern, indem man die Klartextblöcke kürzer wählt, als das eigentlich nötig ist, und die letzten Bits zufällig wählt. Dann ist es praktisch ausgeschlossen, dass zweimal derselbe Block verschlüsselt wird.

Eine andere Möglichkeit, den Low-Exponent-Angriff zu verhindern, besteht darin, größere Verschlüsselungsexponenten zu wählen, die aber immer noch eine effiziente Verschlüsselung zulassen. Üblich ist $e = 2^{16} + 1$ (siehe Übung 8.7).

8.3.7 Auswahl von *d*

Wird bei der Schlüsselerzeugung der öffentliche RSA-Schlüssel zuerst gewählt, zum Beispiel so, wie im vorigen Abschnitt vorgeschlagen, so ensteht ein privater Schlüssel d, der in der Größenordnung von n liegt. Es gibt aber Situationen, in denen es wünschenswert sein könnte, d zuerst zu wählen und dabei möglichst klein zu machen. Wird der private RSA-Schlüssel zum Beispiel auf einer Chipkarte gespeichert, so ist es gut, wenn auch die Entschlüsselung auf dieser Karte stattfindet. Dann muss der private RSA-Schlüssel die Chipkarte nie verlassen und ist so besser geschützt. Chipkarten haben aber keine besonders gute Performanz. Ein kleiner privater RSA-Schlüssel beschleunigt die Entschlüsselung. Es stellt sich aber heraus, dass eine solche Wahl von d nicht sicher ist. D. Boneh und G. Durfee haben nämlich in [17] bewiesen, dass das RSA-Verfahren gebrochen werden kann, wenn $d < n^{0.292}$ ist.

8.3.8 Performanz

Die RSA-Verschlüsselung erfordert eine Exponentiation modulo n. Die Verschlüsselung ist um so effizienter, je kleiner der Exponent ist. Bei kleinen Exponenten muss man aber

Vorkehrungen gegen den Low-Exponent-Angriff treffen. Angriff und Gegenmaßnahmen wurden in Abschn. 8.3.6 beschrieben. Wird zum Beispiel $e = 2^{16} + 1$ gewählt, so benötigt die RSA-Verschlüsselung 16 Quadrierungen und eine Multiplikation modulo n.

Die Entschlüsselung eines RSA-verschlüsselten Textes erfordert ebenfalls eine Exponentiation modulo n. Diesmal ist aber der Exponent d in derselben Größenordnung wie n. Die Verwendung kleiner Entschlüsselungsexponenten ist unsicher, wie in Abschn. 8.3.7 dargestellt wurde. Ist k die Bitlänge von n, so erfordert die Entschlüsselung k Quadrierungen modulo n und $k/2$ Multiplikationen modulo n^d, wenn man davon ausgeht, dass die Hälfte der Bits in der Binärentwicklung von n den Wert 1 haben. Da in der Praxis RSA-Moduln wenigstens die Länge 1024 haben, sind also wenigstens 1024 Quadrierungen und 512 Multiplikationen nötig. Oft werden RSA-Entschlüsselungen auf langsamen Chipkarten durchgeführt. Um eine hinreichende Performanz zu erzielen, werden auf den Chipkarten Koprozessoren eingesetzt, um die Arithmetik modulo n zu beschleunigen.

Die RSA-Entschlüsselung kann mit dem Chinesischen Restsatz beschleunigt werden. Sei $((n, e), d)$ das RSA-Schlüsselpaar von Alice. Sie möchte den Schlüsseltext c entschlüsseln. Sie berechnet

$$m_p = c^d \bmod p, \quad m_q = c^d \bmod q$$

und löst dann die simultane Kongruenz

$$m \equiv m_p \bmod p, \quad m \equiv m_q \bmod q.$$

Dann ist m der ursprüngliche Klartext. Um die simultane Kongruenz zu lösen, berechnet Alice mit dem erweiterten euklidischen Algorithmus ganze Zahlen y_p und y_q mit

$$y_p p + y_q q = 1.$$

Dann setzt sie

$$m = (m_p y_q q + m_q y_p p) \bmod n.$$

Man beachte, dass die Zahlen $y_p p \bmod n$ und $y_q q \bmod n$ nicht von der zu entschlüsselnden Nachricht abhängen, und daher ein für allemal vorberechnet werden können.

Beispiel 8.8 Um die Entschlüsselung aus Beispiel 8.4 zu beschleunigen, berechnet Alice

$$m_p = 119^7 \bmod 11 = 4, \quad m_q = 119^{15} \bmod 23 = 3,$$

sowie $y_p = -2$, $y_q = 1$ und setzt dann

$$m = (4 * 23 - 3 * 2 * 11) \bmod 253 = 26.$$

Wir zeigen, dass Entschlüsseln mit dem chinesischen Restsatz effizienter ist als das Standardverfahren. Dazu machen wir einige teilweise vereinfachende Annahmen. Wir nehmen an, dass der RSA-Modul n eine k-Bit-Zahl ist, dass die Primfaktoren p und q

jeweils $k/2$-Bit-Zahlen sind und dass die Hälfte der Bits in d den Wert 1 hat. Die Multiplikation zweier Zahlen in $\{0,\ldots,n-1\}$ und die anschließende Reduktion des Ergebnisses modulo n benötigt Zeit Ck^2, wobei C eine Konstante ist. Wir verwenden also zur Multiplikation die Schulmethode. Die Berechnung $m = c^d \bmod n$ kostet Zeit $(3/2)Ck^3$,

Bei der Berechnung von m_p und m_q kann wegen des kleinen Satzes von Fermat der Exponent d durch $d_p = d \bmod p-1$ beziehungsweise $d_q = d \bmod q-1$ ersetzt werden. Wir nehmen ebenfalls an, dass die Hälfte der Bits in diesen Exponenten den Wert 1 hat. Dann kostet die Berechnung von m_p und m_q jeweils Zeit $(3/2)C(k/2)^3 = (3/16)k^3$, insgesamt also $(3/8)k^3$. Ignoriert man also die Zeit, die zur Berechnung im chinesischen Restsatz verwendet wird, so hat man eine Beschleunigung um den Faktor 4. Die Anwendung des chinesischen Restsatzes erlaubt eine Vorberechnung und erfordert dann nur noch zwei modulare Multiplikationen. Weil d so groß ist, kann man die Zeit dafür tatsächlich ignorieren. Entschlüsseln mit dem chinesischen Restsatz ist also etwa viermal so schnell wie die Standard-Entschlüsselungsmethode.

8.3.9 Multiplikativität

Sei (n,e) ein öffentlicher RSA-Schlüssel. Werden damit zwei Nachrichten m_1 und m_2 verschlüsselt, so sieht der Angreifer

$$c_1 = m_1^e \bmod n, \quad c_2 = m_2^e \bmod n.$$

Es gilt dann

$$c = c_1c_2 \bmod n = (m_1m_2)^e \bmod n.$$

Kennt der Angreifer also die beiden Klartext-Schlüsseltext-Paare (m_1,c_1) und (m_2,c_2), so kann er $c = c_1c_2 \bmod n$ zu $m = m_1m_2$ entschlüsseln. Die hier beschriebene Multiplikativität von RSA erlaubt Chosen-Ciphertext-Angriffe. Angenommen, ein Angreifer möchte den Chiffretext c entschlüsseln. Er wählt einen Klartext m_1 und verschlüsselt ihn zu

$$c_1 = m_1^e \bmod n.$$

Anschließend berechnet er

$$c_2 = cc_1^{-1} \bmod n.$$

Dabei ist c_1^{-1} das Inverse von c_1 modulo n. Der Angreifer lässt c_2 zu m_2 entschlüsseln und kann so c zu $m = m_1m_2$ entschlüsseln.

8.3.10 Sichere Verwendung

In diesem Abschnitt wurden eine Reihe von Angriffen auf das RSA-Verfahren beschrieben, die selbst dann möglich sind, wenn die RSA-Parameter sicher gewählt sind. Darum

wird in der Praxis *RSA-OAEP* verwendet. Die Abkürzung OAEP steht für *Optimal Asymmetric Encryption*. Diese Variante findet sich im Standard PKCS# 1 [56]. Sie beruht auf dem OAEP-Verfahren von Bellare und Rogaway [8]. Eine Verbesserung stammt von Shoup [70].

Wir erläutern die Funktionsweise von RSA-OAEP. Sei t eine natürliche Zahl mit der Eigenschaft, dass die maximale Laufzeit, die ein Angreiferalgorithmus verbrauchen kann, deutlich kleiner als 2^t ist. Sei k die binäre Länge des RSA-Moduls. Also ist $k \geq 1024$ und sei $l = k - t - 1$. Benötigt werden eine Expansionsfunktion

$$G : \{0,1\}^t \to \{0,1\}^l$$

und eine Kompressionsfunktion

$$H : \{0,1\}^l \to \{0,1\}^t.$$

Diese Funktionen sind öffentlich bekannt. Der Klartextraum ist $\{0,1\}^l$. Soll ein Klartext $m \in \{0,1\}^l$ verschlüsselt werden, so wird zuerst eine Zufallszahl $r \in \{0,1\}^t$ gewählt. Dann wird der Chiffretext

$$c = ((m \oplus G(r)) \circ (r \oplus H(m \oplus G(r))))^e \bmod n$$

berechnet. Bei der Entschlüsselung berechnet der Empfänger zuerst

$$(m \oplus G(r)) \circ (r \oplus H(m \oplus G(r))) = c^d \bmod n.$$

Dann kann er

$$r = (r \oplus H(m \oplus G(r))) \oplus H(m \oplus G(r))$$

und danach

$$m = (m \oplus G(r)) \oplus G(r)$$

berechnen. Der Klartext wird also zu $m \oplus G(r)$ randomisiert. Der Zufallswert r wird zu $(r \oplus H(m \oplus G(r)))$ maskiert.

8.3.11 Verallgemeinerung

Wir erklären, wie man das RSA-Verfahren verallgemeinern kann. Sei G eine endliche Gruppe, in der alle effizient rechnen können. Die Ordnung o dieser Gruppe sei nur Alice bekannt. Alice wählt einen Verschlüsselungsexponenten $e \in \{2, \ldots, o-1\}$, der zu o teilerfremd ist. Ihr öffentlicher Schüssel ist (G, e). Der private Schlüssel ist eine Zahl $d \in \{2, \ldots, o-1\}$, für die $ed \equiv 1 \bmod o$ gilt. Will man eine Nachricht $m \in G$ verschlüsseln, so berechnet man $c = m^e$. Es gilt dann $c^d = m^{ed} = m^{1+ko} = m$ für eine ganze Zahl k.

Auf diese Weise kann c also entschlüsselt werden. Die Funktion

$$G \to G, \quad m \mapsto m^e$$

ist eine sogenannte *Trapdoor-One-Way-Permutation* oder einfach *Trapdoor-Permutation*. Sie ist bijektiv und kann leicht berechnet werden. Sie kann aber nicht in vertretbarer Zeit invertiert werden. Aber die Kenntnis eines Geheimnisses, hier die Gruppenordnung, erlaubt es, diese Funktion effizient zu invertieren.

Das beschriebene Verfahren ist tatsächlich eine Verallgemeinerung des RSA-Verfahrens: Im RSA-Verfahren ist $G = (\mathbb{Z}/n\mathbb{Z})^*$. Die Kenntnis des öffentlichen RSA-Schlüssels (n, e) erlaubt es, effizient in G zu rechnen. Die Ordnung von G ist $o = (p-1)(q-1)$. Ist die Ordnung bekannt, so können, wie in Abschn. 8.3.4 beschrieben, die Faktoren p und q berechnet und damit der private RSA-Schlüssel bestimmt werden. Solange also die Faktorisierung von n nicht bekannt ist, ist auch die Ordnung von $G = (\mathbb{Z}/n\mathbb{Z})^*$ geheim.

Es sind andere Realisierungen dieses Prinzips vorgeschlagen worden, etwa die Verwendung von elliptischen Kurven über $\mathbb{Z}/n\mathbb{Z}$ mit $n = pq$. Die Schwierigkeit, die Ordnung dieser Gruppen zu berechnen, beruht aber in allen Fällen auf der Schwierigkeit, natürliche Zahlen in ihre Primfaktoren zu zerlegen. Es ist eine interessante Frage, ob sich auf andere Weise Gruppen mit geheimer Ordnung erzeugen lassen, die man in einem RSA-ähnlichen Schema einsetzen könnte.

8.4 Das Rabin-Verschlüsselungsverfahren

Es ist sehr vorteilhaft, wenn die Sicherheit eines Verschlüsselungsverfahrens auf die Schwierigkeit eines wichtigen mathematischen Problems zurückgeführt werden kann, das auch ohne die kryptographische Anwendung von Bedeutung ist. Ein solches mathematisches Problem wird nämlich von vielen Mathematikern weltweit bearbeitet. Solange es ungelöst ist, bleibt das Kryptoverfahren sicher. Wird es aber gelöst und wird damit das Kryptosystem unsicher, so wird diese Entdeckung wahrscheinlich schnell bekannt, und man kann entsprechende Vorkehrungen treffen. Bei rein kryptographischen Problemen ist es eher möglich, dass nur ein kleiner Kreis von Personen, z. B. ein Geheimdienst, die Lösung findet und dies den meisten Benutzern verborgen bleibt. Die arglosen Benutzer verwenden dann vielleicht ein unsicheres System in der trügerischen Annahme, es sei sicher.

Die Sicherheit des RSA-Verfahrens hängt zwar mit der Schwierigkeit des Faktorisierungsproblems für natürliche Zahlen zusammen. Es ist aber nicht bekannt, ob das Problem, RSA zu brechen, genauso schwer wie das Faktorisierungsproblem für natürliche Zahlen ist. Anders ist es mit dem Rabin-Verfahren, das nun erklärt wird. Wir werden sehen, dass die Schwierigkeit, das Rabin-Verfahren mit einem Ciphertext-Only-Angriff zu brechen, äquivalent zu einem bestimmten Faktorisierungsproblem ist. Allerdings erweist sich das Verfahren als angreifbar durch Chosen-Cipertext-Angriffe. Zuerst wird aber die Funktionsweise des Rabin-Verfahrens beschrieben.

8.4.1 Schlüsselerzeugung

Alice wählt zufällig zwei Primzahlen p und q mit $p \equiv q \equiv 3 \bmod 4$. Die Kongruenzbedingung vereinfacht und beschleunigt die Entschlüsselung, wie wir unten sehen werden. Aber auch ohne diese Kongruenzbedingung funktioniert das Verfahren. Alice berechnet $n = pq$. Die Primzahlen p und q sind so gewählt, dass der *Rabin-Modul* n eine k-Bit-Zahl ist. Dabei ist k der Sicherheitsparameter. Alices öffentlicher Schlüssel ist der Rabin-Modul n. Ihr geheimer Schlüssel ist (p, q).

8.4.2 Verschlüsselung

Wie beim RSA-Verfahren ist der Klartextraum die Menge $\{0, \ldots, n-1\}$. Um einen Klartext m zu verschlüsseln, besorgt sich Bob den öffentlichen Schlüssel n von Alice und berechnet

$$c = m^2 \bmod n.$$

Der Schlüsseltext ist c.

Wie das RSA-Verfahren kann auch das Rabin-Verfahren zu einer Art Blockchiffre gemacht werden, indem Buchstabenblöcke als Zahlen in der Menge $\{0, 1, \ldots, n-1\}$ aufgefasst werden.

8.4.3 Entschlüsselung

Alice berechnet den Klartext m aus dem Schlüsseltext c durch Wurzelziehen. Dazu geht sie folgendermaßen vor. Sie berechnet

$$m_p = c^{(p+1)/4} \bmod p, \quad m_q = c^{(q+1)/4} \bmod q.$$

Dann sind $\pm m_p + p\mathbb{Z}$ die beiden Quadratwurzeln von $c + p\mathbb{Z}$ in $\mathbb{Z}/p\mathbb{Z}$ und $\pm m_q + q\mathbb{Z}$ die beiden Quadratwurzeln von $c + q\mathbb{Z}$ in $\mathbb{Z}/q\mathbb{Z}$ (siehe Übung 2.21). Die vier Quadratwurzeln von $c + n\mathbb{Z}$ in $\mathbb{Z}/n\mathbb{Z}$ kann Alice mit Hilfe des chinesischen Restsatzes berechnen. Dies funktioniert nach derselben Methode wie die Entschlüsselung eines RSA-Chiffretextes mit dem chinesischen Restsatz. Alice bestimmt mit dem erweiterten euklidischen Algorithmus Koeffizienten $y_p, y_q \in \mathbb{Z}$ mit

$$y_p p + y_q q = 1.$$

Dann berechnet sie

$$r = (y_p p m_q + y_q q m_p) \bmod n, \quad s = (y_p p m_q - y_q q m_p) \bmod n.$$

Man verifiziert leicht, dass $\pm r$, $\pm s$ die vier Quadratwurzeln von c in der Menge $\{0, 1, \ldots, n-1\}$ sind. Eine dieser Quadratwurzeln muss die Nachricht m sein, es ist aber nicht a priori klar, welche.

Beispiel 8.9 Alice verwendet die Primzahlen $p = 11, q = 23$. Dann ist $n = 253$. Bob will die Nachricht $m = 158$ verschlüsseln. Er berechnet

$$c = m^2 \bmod n = 170.$$

Alice berechnet $y_p = -2$, $y_q = 1$ wie in Beispiel 8.8. Sie ermittelt die Quadratwurzeln

$$m_p = c^{(p+1)/4} \bmod p = c^3 \bmod p = 4,$$
$$m_q = c^{(q+1)/4} \bmod q = c^6 \bmod q = 3.$$

Sie bestimmt

$$r = (y_p p m_q + y_q q m_p) \bmod n = -2 * 11 * 3 + 23 * 4 \bmod n = 26$$

und

$$s = (y_p p m_q - y_q q m_p) \bmod n = -2 * 11 * 3 - 23 * 4 \bmod n = 95.$$

Die Quadratwurzeln von 170 mod 253 in $\{1, \ldots, 252\}$ sind $26, 95, 158, 227$. Eine dieser Quadratwurzeln ist der Klartext.

Es gibt verschiedene Methoden, aus den vier Quadratwurzeln den richtigen Klartext auszusuchen. Alice kann einfach diejenige Nachricht auswählen, die ihr am wahrscheinlichsten erscheint. Das ist aber nicht besonders treffsicher und möglicherweise wählt Alice die falsche Nachricht aus. Es ist auch möglich, den Klartexten eine spezielle Struktur zu geben. Dann Wählt Alice also die Quadratwurzel aus, die die betreffende Struktur aufweist. Man kann z. B. nur Klartexte m zulassen, in denen die letzten 64 Bit gleich den vorletzten 64 Bit sind. Verwendet man aber das Rabin-Verfahren in dieser Art, so kann man die Äquivalenz zum Faktorisierungsproblem nicht mehr beweisen.

8.4.4 Effizienz

Rabin-Verschlüsselung erfordert nur eine Quadrierung modulo n. Das ist sogar effizienter als die RSA-Verschlüsselung mit dem Exponenten 3, bei der eine Quadrierung und eine Multiplikation mod n nötig ist. Die Entschlüsselung erfordert je eine modulare Exponentiation mod p und mod q und die Anwendung des chinesischen Restsatzes. Das entspricht dem Aufwand, der zur RSA-Entschlüsselung bei Anwendung des chinesischen Restsatz nötig ist.

8.4.5 Sicherheit

Wir zeigen, dass Angreifer, die die Fähigkeit, haben, Rabin zu entschlüsseln, den RSA-Modul faktorisieren können.

Angenommen, es gibt einen erfolgreichen Ciphertext-Only-Angriff. Der Angreifer kann also zu jedem Quadrat $c + n\mathbb{Z}$ eine Quadratwurzel $m + n\mathbb{Z}$ bestimmen, sagen wir mit einem Algorithmus A. Der Algorithmus A liefert bei Eingabe von $c \in \{0, 1, \ldots, n-1\}$ eine ganze Zahl $m \leftarrow A(c)$, $m \in \{0, 1, \ldots, n-1\}$, für die die Restklasse $m + n\mathbb{Z}$ eine Quadratwurzel von $c + n\mathbb{Z}$ ist.

Um n zu faktorisieren, wählt Oskar zufällig und gleichverteilt eine Zahl $x \in \{1, \ldots, n-1\}$. Wenn $\gcd(x, n) \neq 1$ ist, hat Oskar den Modul n faktorisiert. Andernfalls berechnet er

$$c = x^2 \bmod n \text{ und } m \leftarrow A(c).$$

Die Restklasse $m + n\mathbb{Z}$ ist eine der Quadratwurzeln von $c + n\mathbb{Z}$. Sie muss nicht mit $x + n\mathbb{Z}$ übereinstimmen. Aber m erfüllt eines der folgenden Bedingungspaare

$$m \equiv x \bmod p \text{ und } m \equiv x \bmod q, \tag{8.6}$$

$$m \equiv -x \bmod p \text{ und } m \equiv -x \bmod q, \tag{8.7}$$

$$m \equiv x \bmod p \text{ und } m \equiv -x \bmod q, \tag{8.8}$$

$$m \equiv -x \bmod p \text{ und } m \equiv x \bmod q. \tag{8.9}$$

Im Fall (8.6) ist $m = x$ und $\gcd(m - x, n) = n$. Im Fall (8.7) ist $m = n - x$ und $\gcd(m-x, n) = 1$. Im Fall (8.8) ist $\gcd(m-x, n) = p$. Im Fall (8.9) ist $\gcd(m-x, n) = q$. Da x zufällig gewählt wurde, tritt jeder dieser Fälle mit derselben Wahrscheinlichkeit auf. Daher wird bei einem Durchlauf des Verfahrens die Zahl n mit Wahrscheinlichkeit $\geq 1/2$ faktorisiert, und bei k Durchläufen des Verfahrens wird n mit Wahrscheinlichkeit $\geq 1 - (1/2)^k$ zerlegt.

Beispiel 8.10 Wie in Beispiel 8.9 ist $n = 253$. Angenommen, ein Angreifer kann mit dem Algorithmus A Quadratwurzeln modulo 253 bestimmen. Er wählt $x = 17$ und stellt fest, dass $\gcd(17, 253) = 1$ ist. Als nächstes bestimmt Oskar $c = 17^2 \bmod 253 = 36$. Die Quadratwurzeln von 36 mod 253 sind $6, 17, 236, 247$. Es ist $\gcd(6 - 17, n) = 11$ und $\gcd(247 - 17, 253) = 23$. Wenn A also eine dieser beiden Quadratwurzeln berechnet, hat der Angreifer die Faktorisierung von n gefunden.

Im beschriebenen Faktorisierungsverfahren wurde vorausgesetzt, dass der Klartextraum aus allen Zahlen in der Menge $\{0, 1, \ldots, n-1\}$ besteht. Die Argumentation ist nicht mehr richtig, wenn der Klartextraum wie in Abschn. 8.4.3 auf Klartexte mit bestimmter Struktur eingeschränkt wird. Dann kann der Algorithmus A nämlich nur Verschlüsselungen dieser speziellen Klartexte entschlüsseln. Der Angreifer muss dann x mit dieser speziellen Struktur wählen. Die anderen Quadratwurzeln von $x^2 + n\mathbb{Z}$ haben mit hoher Wahrscheinlichkeit nicht die spezielle Struktur. Darum liefert A auch nur die Quadratwurzel x, die Oskar ohnehin schon kannte.

Folgt aus der Argumentation dieses Abschnitts, dass das Rabin-Verfahren sicher ist? Leider nicht! Im nächsten Abschnitt erklären wir nämlich einen erfolgreichen Chosen-Chipertext-Algorithmus gegen das Rabin-Verfahren.

8.4.6 Ein Chosen-Ciphertext-Angriff

Wir haben gesehen, dass ein Angreifer natürliche Zahlen faktorisieren kann, wenn er das Rabin-Verfahren brechen kann. Man kann dies als Sicherheitsvorteil ansehen. Andererseits erlaubt dieser Umstand aber auch einen Chosen-Ciphertext-Angriff.

Angenommen, der Angreifer kann einen selbst gewählten Schlüsseltext entschlüsseln. Dann wählt er $x \in \{1, \ldots, n-1\}$ zufällig und berechnet den Schlüsseltext $c = x^2 \bmod n$. Diesen Schlüsseltext entschlüsselt er anschließend. Wie wir in Abschn. 8.4.5 gesehen haben, kann der Angreifer danach den Modul n mit Wahrscheinlichkeit $1/2$ faktorisieren.

Um den Chosen-Ciphertext-Angriff zu verhindern, kann man, wie in 8.4.3 beschrieben, den Klartextraum auf Klartexte mit bestimmter Struktur beschränken.

Einige Angriffe, die beim RSA-Verfahren möglich sind, kann man auch auf das Rabin-Verfahren anwenden. Dies gilt insbesondere für den Low-Exponent-Angriff, den Angriff, der bei zu kleinem Klartextraum möglich ist, und die Ausnutzung der Multiplikativität. Die Details werden in den Übungen behandelt.

8.4.7 Sichere Verwendung

Will man das Rabin-Verfahren sicher machen, muss man es so verwenden, wie das in Abschn. 8.3.10 beschrieben wurde.

8.5 Sicherheitsmodelle

Sicherheitsmodelle für symmetrische Verschlüsselungsverfahren wurden in Kap. 4 behandelt. In diesem Abschnitt beschreiben wir analoge Modelle für Public-Key-Verfahren. Chosen-Plaintext-Angriffe sind allerdings die einfachsten Angriffe auf Public-Key-Kryptosystem. Alle Angreifer kennen nämlich die öffentlichen Schlüssel und können damit Klartexte ihrer Wahl verschlüsseln. Darum fallen bei Public-Key-Verfahren die Modelle der semantischen Sicherheit und der Chosen-Plaintext-Sicherheit zusammen.

8.5.1 Chosen-Plaintext-Sicherheit

Das Modell der *Chosen-Plaintext-Sicherheit* hat viele Ähnlichkeiten mit dem entsprechenden Modell für symmetrische Chiffren (siehe Abschn. 4.4). Chosen-Plaintext-Sicherheit wird auch *CPA-Sicherheit*, *semantische Sicherheit* oder *Indistinguishability-Under-Chosen-Plaintext-Attack (IND-CPA)*genannt.

Ein CPA-Angreifer auf ein Public-Key-Kryptosystem

$$\mathbf{E} = (\mathbf{K}, \mathbf{P}, \mathbf{C}, \mathbf{KeyGen}, \mathbf{Enc}, \mathbf{Dec})$$

ist ein probabilitischer Algorithmus A. Seine Eingabe ist ein Sicherheitsparameter $k \in \mathbb{N}$ und ein öffentlicher Schlüssel e aus der Menge $\mathbf{Pub}(k)$. Ziel des Angreifers ist es, zwischen Wahrscheinlichkeitsverteilungen von Chiffretexten zu zwei unterschiedlichen, gleich langen Klartexten zu unterscheiden. Wie im entsprechenden Modell für symmetrische Chiffren erzeugt der Angreifer dieses Klartextpaar selbst. Der Angreifer bekommt die Verschlüsselung eines der beiden Klartexte und soll entscheiden, welcher Klartext verschlüsselt wurde. Den entsprechenden Chiffretext erhält der Angreifer auch hier von einem Orakel, auf das er einmal zugreifen darf. Dieses Orakel ist $\mathbf{Enc}_{b,e}$. Dabei ist b ein Bit und e der gewählte öffentliche Schlüssel aus $\mathbf{Pub}(k)$. Erhält das Orakel ein Paar (P_0, P_1) von Klartexten gleicher Länge, so gibt es den Chiffretext $C \leftarrow \mathbf{Enc}(e, P_b)$ zurück. Ist also $b = 0$, verschlüsselt das Orakel den linken Klartext des Paares (P_0, P_1). Ist $b = 1$, verschlüsselt das Orakel den rechten Klartext dieses Paares. Um zu zeigen, dass der Angreifer Zugriff auf das Orakel $\mathbf{Enc}_{b,e}$ hat, schreiben wir $A^{\mathbf{Enc}_{b,e}}$. Der Angreifer erzeugt also ein Paar (P_0, P_1) von Klartexten gleicher Länge und ruft das Orakel mit diesem Paar als Eingabe auf. Das Orakel gibt die Verschlüsselung von P_b zurück. Das Orakel modelliert alle Möglichkeiten des Angreifers, an die Verschlüsselung von P_0 oder P_1 zu kommen. Anschließend versucht der Angreifer, b zu bestimmen. Er gibt also ein Bit b' zurück. Ist $b = b'$, so hat der Angreifer Erfolg und andernfalls nicht. Die Bedingung, dass die beiden Klartexte gleiche Länge haben müssen, ist plausibel, weil Klartext- und Chiffretextlängen typischerweise korreliert sind und darum Wahrscheinlichkeitsverteilungen auf Schlüsseltexten zu Klartexten unterschiedlicher Länge immer leicht unterscheidbar sind. Das zeigt Beispiel 4.2. Auch der Vorteil des Angreifers wird wie in Abschn. 4.4 definiert. Dazu wird Experiment 8.1 verwendet.

Experiment 8.1 ($\mathrm{Exp}^{\mathrm{CPA}}_{\mathbf{E}}(A, k)$)

(Experiment, das über den Erfolg eines Angreifer auf die semantische Sicherheit des Public-Key-Kryptosystems **E** entscheidet)

$b \xleftarrow{\$} \{0, 1\}$
$(e, d) \leftarrow \mathbf{KeyGen}(1^k)$
$b' \leftarrow A^{\mathbf{Enc}_{b,K}}(1^k, e)$
if $b = b'$ **then**
  return 1
else
  return 0
end if

Für $k \in \mathbb{N}$ ist der Vorteil eines Angreifers A gegen das Public-Key-Kryposystem **E**

$$\mathbf{Adv}^{\mathrm{CPA}}_{\mathbf{E}}(A, k) = 2\Pr[\mathrm{Exp}^{\mathrm{CPA}}_{\mathbf{E}}(A, k) = 1] - 1. \tag{8.10}$$

Die Laufzeit von CPA-Angreifern ist genauso definiert, wie die Laufzeit von Angreifern gegen die semantische Sicherheit oder die CPA-Sicherheit von symmetrischen Kryptosystemen. Damit können wir jetzt die CPA-Sicherheit von Public-Key-Verschlüsselungsverfahren definieren.

Definition 8.2 Seien $T : \mathbb{N} \to \mathbb{N}$ und $\varepsilon : \mathbb{N} \to [0, 1]$ Funktionen. Ein Public-Key-Verschlüsselungsverfahren **E** heißt (T, ε)-sicher gegen Chosen-Plaintext-Angriffe, wenn für alle $k \in \mathbb{N}$ und alle Chosen-Plaintext-Angreifer A, die höchstens Laufzeit $T(k)$ haben, gilt: $Adv_{\mathbf{E}}^{\mathrm{CPA}}(A, k) < \varepsilon(k)$.

Wir definieren auch asymptotische Sicherheit von Public-Key-Verschlüsselungsverfahren.

Definition 8.3 Ein Public-Key-Verschlüsselungsverfahren **E** ist asymptotisch sicher gegen Chosen-Plaintext-Angriffe, wenn für alle $c > 0$ und alle polynomiell beschränkten CPA-Angreifer A gilt: $Adv_{\mathbf{E}}^{\mathrm{CPA}}(A, k) = \mathrm{O}(1/k^c)$.

Weil für Public-Key-Verfahren Chosen-Plaintext-Sicherheit und semantische Sicherheit dasselbe sind, nennt man Public-Key-Kryptosysteme, die gegen Chosen-Plaintext-Angriffe sicher sind auch *semantisch sicher*. Auch die Bezeichnung *IND-CPA-sicher* wird verwendet. Sie steht für *Indistinguishability under Chosen Plaintext Attacks* und bezieht sich darauf, dass Angreifer die Verschlüsselung unterschiedlicher Klartexte nicht unterscheiden können.

8.5.2 Chosen-Ciphertext-Sicherheit

IND-CPA-Sicherheit ist nicht das stärkste Sicherheitsmodell. Wie bei symmetrischen Verschlüsselungsverfahren sind auch hier Chosen-Ciphertext-Angriffe möglich. Ein solcher Angriff auf das RSA-Verfahren wird zum Beispiel von Bleichenbacher in [15] beschrieben. Eine Formalisierung von IND-CCA-Angriffen wird dem Leser als Übung überlassen.

Es kann gezeigt werden, dass IND-CCA-sichere Public-Key-Verschlüsselungssysteme, einem Angreifer auch keine Möglichkeit bieten, den Schlüsseltext so zu verändern, dass sich der Klartext in kontrollierter Weise ändert. Diese Eigenschaft heißt *Non-Malleability*.

8.5.3 Sicherheitsbeweise

Heute sind keine nachweisbar schwierigen mathematischen Probleme bekannt, die zur Konstruktion von Public-Key-Verschlüsselungsverfahren verwendet werden können. Daher gibt es auch keine beweisbar sicheren Public-Key-Verschlüsselungsverfahren. Statt

dessen werden *Sicherheitsreduktionen* verwendet. In einer solchen Reduktion wird bewiesen, dass Polynomzeit-Angreifer auf das Public-Key-Kryptosystem genutzt werden können, um Polynomzeitalgorithmen zur Lösung von Berechnungsproblemen zu konstruieren, die in Wirklichkeit als schwer angesehen werden. Zum Beispiel konnte in Abschn. 8.4.5 ein Rabin-Angreifer, der entschlüsseln kann, zur Konstruktion eines Faktorisierungsalgorithmus verwendet werden. Solange die zugrundeliegenden Berechnungsprobleme also schwer sind, kann es solche Polynomzeit-Angreifer nicht geben. Sicherheitsreduktionen erlauben es also, die Sicherheit von kryptographischen Verfahren auf die Schwierigkeit von Berechnungsproblemen zurückzuführen. Warum ist das ein gutes Argument? Wenn die Berechnungsprobleme „prominent“ und gut untersucht sind wie zum Beispiel das Faktorisierungsproblem, erhöht das (vielleicht) die Wahrscheinlichkeit, dass sie schwer bleiben oder zumindest schnell bekannt würde, wenn effiziente Algorithmen zu ihrer Lösung gefunden werden. Im letzteren Fall könnte man die unsicheren Verfahren schnell ersetzen (wenn es Ersatz gibt).

Die Aussagekraft von Sicherheitsreduktionen wird aber dadurch eingeschränkt, dass sie oft nicht explizit sind, sondern die asymptotischen Notationen O oder o verwenden. Dann kann kein Zusammenhang zwischen schwierigen Instanzen der zugrundeliegenden Berechnungsprobleme und sicheren Parametern des Public-Key-Kryptosystems hergestellt werden. Es wäre aber wünschenswert, Aussagen von der Art: „das Verfahren hat 100-Bit-Sicherheit wenn 1024-Bit RSA-Moduln verwendet werden“ beweisen zu können. Das geht nur, wenn die Reduktion explizit ist.

Bellare und Rogaway konnten in [8] zeigen, dass die Chosen-Ciphertext-Sicherheit des RSA-OAEP-Verfahrens auf die Schwierigkeit des *RSA-Problems* reduziert werden kann. Das RSA-Problem besteht darin, für einen k-Bit RSA-Modul n und einen öffentlichen Schlüssel $e \in \mathbf{Pub}(k)$ die Permutation

$$\mathbb{Z}_n \to \mathbb{Z}_n, \quad m \mapsto m^e \mod n \tag{8.11}$$

zu invertieren. Die Reduktion ist explizit, erlaubt es also, einen Zusammenhang zwischen den Instanzen des RSA-Problems und den RSA-Parametern herzustellen. Allerdings wird im Beweis das *Random-Oracle-Modell* verwendet wird. In diesem Modell geht man davon aus, dass die Expansionsfunktion G und die Kompressionsfunktion H zufällig aus der jeweiligen Menge aller solcher Funktionen gewählt wurde. Dies ist keine realistische Annahme, weil eine solche zufällige Wahl unmöglich ist. Es gibt eine riesige Zahl solcher Funktionen und es gibt keine Möglichkeit, sie in effizienter Weise darzustellen. Daher erlaubt der Reduktionsbeweis nicht die Bestimmung sicherer RSA-Parameter. Die Aussagekraft des Sicherheitsbeweises von Bellare und Rogaway wird auch dadurch eingeschränkt, dass sie die Sicherheit des RSA-OAEP-Verfahrens nicht auf das Faktorisierungsproblem für RSA-Moduln sondern auf das RSA-Problem reduzieren, das weniger prominent ist. Trotzdem ist RSA-OAEP von großer praktischer Bedeutung weil das ursprüngliche RSA-Verfahren leicht angreifbar ist.

8.6 Diffie-Hellman-Schlüsselaustausch

In diesem Abschnitt beschreiben wir das Verfahren von Diffie und Hellman, geheime Schlüssel über unsichere Leitungen auszutauschen. Das Diffie-Hellman-Verfahren ist zwar kein Public-Key-Verschlüsselungsverfahren, aber es ist die Grundlage des ElGamal-Verfahrens, das als nächstes behandelt wird.

Die Situation ist folgende: Alice und Bob wollen mit einem symmetrischen Verschlüsselungsverfahren kommunizieren. Sie sind über eine unsichere Leitung verbunden und haben noch keinen Schlüssel ausgetauscht. Das Diffie-Hellman-Verfahren erlaubt es Alice und Bob, einen geheimen Schlüssel über die öffentliche, nicht gesicherte Leitung auszutauschen, ohne dass ein Zuhörer den Schlüssel erfährt.

Die Sicherheit des Diffie-Hellman-Verfahrens beruht nicht auf dem Faktorisierungsproblem für natürliche Zahlen, sondern auf einem anderen zahlentheoretischen Problem, das wir hier kurz vorstellen.

8.6.1 Diskrete Logarithmen

Sei p eine Primzahl. Wir wissen, dass die prime Restklassengruppe mod p zyklisch ist von der Ordnung $p-1$. Sei g eine Primitivwurzel mod p. Dann gibt es für jede Zahl $A \in \{1, 2, \ldots, p-1\}$ genau einen Exponenten $a \in \{0, 1, 2, \ldots, p-2\}$ mit

$$A \equiv g^a \bmod p.$$

Dieser Exponent a heißt *diskreter Logarithmus* von A zur Basis g. Wir schreiben $a = \log_g A$. Die Berechnung solcher diskreter Logarithmen gilt als schwieriges Problem. Bis heute sind keine effizienten Algorithmen bekannt, die dieses Problem lösen.

Beispiel 8.11 Sei $p = 13$. Eine Primitivwurzel modulo 13 ist 2. In der folgenden Tabelle finden sich die diskreten Logarithmen aller Elemente der Menge $\{1, 2, \ldots, 12\}$ zur Basis 2.

A	1	2	3	4	5	6	7	8	9	10	11	12
$\log_2 A$	0	1	4	2	9	5	11	3	8	10	7	6

Diskrete Logarithmen kann man auch in anderen zyklischen Gruppen definieren. Sei G eine endliche zyklische Gruppe der Ordnung n mit Erzeuger g und sei A ein Gruppenelement. Dann gibt es einen Exponenten $a \in \{0, 1, \ldots, n-1\}$ mit

$$A = g^a.$$

Dieser Exponent a heißt dann auch *diskreter Logarithmus* von A zur Basis g. Wir werden sehen, dass sich das Diffie-Hellman-Verfahren in allen Gruppen G sicher implementieren lässt, in denen die Berechnung diskreter Logarithmen schwer und die Ausführung der Gruppenoperationen leicht ist.

Beispiel 8.12 Betrachte die additive Gruppe $\mathbb{Z}/n\mathbb{Z}$ für eine natürliche Zahl n. Sie ist zyklisch von der Ordnung n. Ein Erzeuger dieser Gruppe ist $1+n\mathbb{Z}$. Sei $A \in \{0, 1, \ldots, n-1\}$. Der diskrete Logarithmus a von $A + n\mathbb{Z}$ zur Basis $1 + n\mathbb{Z}$ genügt der Kongruenz

$$A \equiv a \bmod n.$$

Also ist $a = A$. Die anderen Erzeuger von $\mathbb{Z}/n\mathbb{Z}$ sind die Restklassen $g + n\mathbb{Z}$ mit $\gcd(g, n) = 1$. Der diskrete Logarithmus a von $A + n\mathbb{Z}$ zur Basis $g + n\mathbb{Z}$ genügt der Kongruenz

$$A \equiv ga \bmod n.$$

Diese Kongruenz kann man mit dem erweiterten euklidischen Algorithmus lösen. In $\mathbb{Z}/n\mathbb{Z}$ gibt es also ein effizientes Verfahren zur Berechnung diskreter Logarithmen. Daher ist diese Gruppe für die Implementierung des Diffie-Hellman-Verfahrens ungeeignet.

8.6.2 Schlüsselaustausch

Das Diffie-Hellman-Verfahren funktioniert folgendermaßen: Bob und Alice wollen sich auf einen Schlüssel K einigen, haben aber nur eine Kommunikationsverbindung zur Verfügung, die abgehört werden kann. Bob und Alice einigen sich auf eine Primzahl p und eine Primitivwurzel g modulo p in $\mathbb{Z}_p$. Statt einer Primimitiwurzel kann auch eine ganze Zahl g gewählt werden, die modulo p hinreichend hohe Ordnung hat. Die Auswahl von p und g wird in Abschn. 8.6.4 diskutiert. Die Primzahl p und die Primitivwurzel g können öffentlich bekannt sein.

Alice wählt eine natürliche Zahl $a \in \{0, 1, \ldots, p-2\}$ zufällig. Sie berechnet

$$A = g^a \bmod p$$

und schickt das Ergebnis A an Bob. Aber sie hält den Exponenten a geheim. Bob wählt eine natürliche Zahl $b \in \{0, 1, \ldots, p-2\}$ zufällig. Er berechnet

$$B = g^b \bmod p$$

und schickt das Ergebnis an Alice. Den Exponenten b hält er geheim. Um den geheimen Schlüssel zu berechnen, berechnet Alice

$$B^a \bmod p = g^{ab} \bmod p,$$

und Bob berechnet

$$A^b \bmod p = g^{ab} \bmod p.$$

Der gemeinsame Schlüssel ist

$$K = g^{ab} \bmod p.$$

Beispiel 8.13 Sei $p = 17$, $g = 3$. Alice wählt den Exponenten $a = 7$, berechnet $g^a \bmod p = 11$ und schickt das Ergebnis $A = 11$ an Bob. Bob wählt den Exponenten $b = 4$, berechnet $g^b \bmod p = 13$ und schickt das Ergebnis $B = 13$ an Alice. Alice berechnet $B^a \bmod p = 4$. Bob berechnet $A^b \bmod p = 4$. Also ist der ausgetauschte Schlüssel 4.

8.6.3 Das Diffie-Hellman-Problem

Ein Lauscher an der unsicheren Leitung erfährt die Zahlen p, g, A und B. Er erfährt aber nicht die diskreten Logarithmen a von A und b von B zur Basis g. Er muss den geheimen Schlüssel $K = g^{ab} \bmod p$ berechnen. Das nennt man das *Diffie-Hellman-Problem (DH)*. Es besteht also darin, bei Eingabe von p, g, A, B die Zahl $K = g^{ab} \bmod p$ zu bestimmen. Um dieses Problem von dem weiter unten besprochenen Decisional-Diffie-Hellman-Problem abzugrenzen, nennt man es auch *Computational-Diffie-Hellman-Problem (CDH)*.

Wer diskrete Logarithmen mod p berechnen kann, ist in der Lage, das Diffie-Hellman-Problem zu lösen. Das ist auch die einzige bekannte allgemein anwendbare Methode, um das Diffie-Hellman-Verfahren zu brechen. Es ist aber nicht bewiesen, dass das tatsächlich die einzige Methode ist, ob also jemand, der das Diffie-Hellman-Problem effizient lösen kann, auch diskrete Logarithmen effizient berechnen kann.

Solange das Diffie-Hellman-Problem nicht in vertretbarer Zeit lösbar ist, ist es für einen Angreifer unmöglich, den geheimen Schlüssel zu bestimmen. Soll es aber für den Angreifer unmöglich sein, aus der öffentlich verfügbaren Information irgendwelche Informationen über den transportierten Schlüssel zu gewinnen, muss das *Decisional-Diffie-Hellman-Problem (DDH)* unangreifbar sein (siehe [16]). Dieses Problem lässt sich folgendermaßen beschreiben. Ein Angreifer erhält drei Zahlen: $A = g^a \bmod p$, $B = g^b \bmod p$, $C = g^c \bmod p$. Dabei wurden entweder a, b, c zufällig und gleichverteilt in $\{0, \ldots, p-2\}$ gewählt oder $c = ab \bmod (p-1)$ gesetzt. Im zweiten Fall heißt (A, B, C) *Diffie-Hellman-Tripel*. Der Angreifer muss entscheiden, ob ein solches Tripel vorliegt oder nicht. Kann er das nicht, ist es ihm unmöglich, aus g^a und g^b Rückschlüsse auf g^{ab} zu ziehen. Wer das Diffie-Helman-Problem lösen kann, ist offensichtlich auch dazu in der Lage, das Decisional-Diffie-Hellman-Problem lösen. Umgekehrt ist das nicht klar. Tatsächlich erscheint DDH in der primen Restklassengruppe modulo einer Primzahl p leichter als CDH. Der Grund dafür besteht in folgender einfachen Beobachtung.

Theorem 8.5 *Sei p eine Primzahl, sei g eine Primitivwurzel modulo p und seien $a, b \in \{0, \ldots, p-2\}$. Dann ist g^{ab} genau dann ein quadratischer Rest modulo p, wenn g^a oder g^b ein quadratischer Rest ist modulo p.*

Beweis Das Theorem folgt daraus, dass eine Potenz von g genau dann ein quadratischer Rest modulo p ist, wenn der Exponent gerade ist. □

DDH-Angreifer 8.1 nutzt Theorem 8.5 aus. Die Idee ist, dass der Angreifer prüft, ob das Kriterium aus Theorem 8.5 erfüllt ist. Wenn es erfüllt ist, gibt er 1 zurück. Das bedeutet, dass er das Tripel für ein Diffie-Hellman-Tripel hält. Andernfalls gibt er 0 zurück. Er glaubt also, dass kein Diffie-Hellman-Tripel vorliegt. Damit liegt der Angreifer nicht immer richtig, aber, wie wir zeigen werden, häufig.

Angreifer 8.1 ($A(p, g, A, B, C)$)

(DDH-Unterscheider)

if $\left(\left(\frac{A}{p}\right) = 1 \text{ oder } \left(\frac{B}{p}\right) = 1\right) \text{ und } \left(\frac{C}{p}\right) = 1$ **then**
  return 1
else
  return 0
end if

Die Wirksamkeit des Angreifers wird im Experiment 8.2 überprüft.

Experiment 8.2 ($\text{Exp}_{p,g}^{\text{DDH}}(A)$)

(Experiment, das über den Erfolg eines DDH-Unterscheiders A entscheidet)

$a \xleftarrow{\$} \{0, \ldots, p-2\}$
$A \leftarrow g^a \bmod p$
$b \xleftarrow{\$} \{0, \ldots, p-2\}$
$B \leftarrow g^b \bmod p$
$x \xleftarrow{\$} \{0, 1\}$
if $x = 0$ **then**
  $c \xleftarrow{\$} \{0, \ldots, p-2\}$
else
  $c \leftarrow ab \bmod (p-1)$
end if
$C \leftarrow g^c \bmod p$
$x' \leftarrow A(p, g, A, B, C)$

if $x = x'$ **then**
  return 1
else
  return 0
end if

Im Experiment 8.2 können Angreifer immer raten. Dann ist ihre Erfolgswahrscheinlichkeit $1/2$. Der Vorteil eines DDH-Unterscheiders A ist darum

$$\mathbf{Adv}_{p,g}(A) = 2\Pr[\mathrm{Exp}^{\mathrm{DDH}}_{p,g}(A) = 1] - 1. \tag{8.12}$$

Theorem 8.6 *Der Vorteil des DDH-Angreifers* 8.1 *ist* $1/2$.

Beweis Wenn im Experiment $x = 1$ gewählt wird, dann gibt der Unterscheider die richtige Antwort. Dies geschieht mit Wahrscheinlichkeit $1/2$. Angenommen, $x = 0$ wird gewählt. Wir analysieren die Wahrscheinlichkeit, mit der der Algorithmus in diesem Fall falsch antwortet. Dies ist in Tab. 8.3 dargestellt. Hier steht „R“ für einen quadratischen Rest und „N“ für einen quadratischen Nichtrest.

Jede Zeile in dieser Tabelle tritt mit Wahrscheinlichkeit $1/16$ auf. Mit Wahrscheinlichkeit $1/4$ wählt der Angreifer also $x = 0$ und gibt dann die falsche Antwort. Der Angreifer antwortet also mit Wahrscheinlichkeit $3/4$ richtig. Sein Vorteil ist $1/2$. □

8.6.4 Auswahl von p

Wie sollen die Primzahl p und die Basiszahl g gewählt werden? Die Primzahl p muss so gewählt werden, dass es unmöglich ist, mit den heute bekannten Algorithmen diskrete Logarithmen modulo p zu berechnen. Algorithmen zur Berechnung von diskreten Logarithmen werden in Kap. 10 behandelt. Eine Übersicht über die Mindestgrößen von

Tab. 8.3 Antworten des DDH-Angreifers

A	B	C	Antwort
R	R	R	falsch
R	R	NR	richtig
R	NR	R	falsch
R	NR	NR	richtig
NR	R	R	falsch
NR	R	NR	richtig
NR	NR	R	richtig
NR	NR	NR	falsch

Tab. 8.4 Mindestgröße von Diffie-Hellman-Primzahlen

Schutz bis	Mindestgröße
2015	1248
2020	1776
2030	2432
2040	3248
für absehbare Zukunft	15.424

Tab. 8.5 Mindestgröße von Untergruppen von Primzahlordnung

Schutz bis	Bitlänge von q
2015	160
2020	192
2030	224
2040	256
für absehbare Zukunft	512

Primzahlen in Abhängigkeit von der erwarteten Schutzdauer findet sich in Tab. 8.4. Sie wurde im Projekt ECRYPT II ermittelt (siehe [73]).

Es ist außerdem nötig, p so zu wählen, dass die bekannten DL-Algorithmen kein leichtes Spiel haben. Man muss z. B. vermeiden, dass $p-1$ nur kleine Primfaktoren hat. Sonst können diskrete Logarithmen mod p nämlich mit dem Algorithmus von Pohlig-Hellman (siehe Abschn. 10.5) berechnet werden. Man muss auch vermeiden, dass p für das Zahlkörpersieb besonders geeignet ist. Dies ist der Fall, wenn es ein irreduzibles Polynom vom Grad 5 gibt, das sehr kleine Koeffizienten und eine Nullstelle mod p hat.

Nun zur Auswahl von g. Die erste Idee war, g als Primitivwurzel modulo p zu wählen. Das ist im Allgemeinen aber schwer. Wählt man aber spezielle Primzahlen, etwa $p = 2q + 1$ mit einer Primzahl q, so lässt sich eine Primitivwurzel modulo p leicht finden (siehe Abschn. 2.22).

In Abschn. 8.6.3 wurde aber gezeigt, dass bei einer Auswahl von g als Primitivwurzel das Decisional-Diffie-Hellman-Problem angegriffen werden kann. Wählt man statt dessen g so, dass die Restklasse von g modulo p Primzahlordnung q hat mit einer hinreichend großen Primzahl q, dann gilt DDH nach heutiger Auffassung als schwierig. Mindestgrößen für q mit der entsprechenden Schutzdauer finden sich in Tab. 8.5.

Wie findet man ein solches p und g? Das macht Algorithmus 8.1. Bei Eingabe zweier Sicherheitsparameter $k_1, k_2 \in \mathbb{N}, k_1 > k_2$ berechnet er eine k_1-Bit Primzahl von der Form $p = mq + 1$. Dabei ist q eine k_2-Bit-Primzahl und m ist eine natürliche Zahl. Außerdem berechnet Algorithmus 8.1 findDHParameters eine Basiszahl g, deren Restklasse modulo p die Ordnung q hat. Mindestgrößen für k_1 und k_2 finden sich in den Tab. 8.4 und 8.5

Algorithmus 8.1 (findDHParameters(k_1, k_2))

(Erzeugung sicherer Parameter für den Diffie-Hellman-Schlüsselaustausch)

repeat
  Wähle zufällige k_2-Bit-Primzahl q
  Wähle zufällig eine natürliche $(k_1 - k_2)$-Bit-Zahl m
until $p = mq + 1$ ist k_1-Bit-Primzahl
repeat
  $x \xleftarrow{\$} \{2, \ldots, p-2\}$
  $g \leftarrow x^m \bmod p$
until $g \not\equiv 1 \bmod p$
return (p, q, g)

Die Korrektheit des Algorithmus sieht man so: Nach Theorem 2.12 ist die Ordnung der Restklasse von g ein Teiler von $p-1 = mq$. Nach Theorem 2.10 ist die Ordnung von g^m modulo p also entweder q oder 1. Der Test im Algorithmus zeigt, welcher Fall vorliegt.

Die Analyse der Laufzeit von Algorithmus SecureDHParameters nehmen wir nicht vor. Dazu müsste man zum Beispiel die Anzahl der Zahlen von der Form $mq + 1$ untersuchen, die Primzahlen sind. Dies ist ein schwieriges Problem der analytischen Zahlentheorie. Wir merken aber an, dass die Wahl von x erfolgreich ist, wenn x eine Primitivwurzel modulo p ist. Nach Theorem 2.11 ist die Anzahl der Primitivwurzeln in der Menge $\{2, \ldots, p-2\}$ mindestens $\varphi(p-1)$ und in [62] wird gezeigt, dass $\varphi(p-1) \geq (p-1)/(6 \ln \ln(p-1))$ gilt.

8.6.5 Man-In-The-Middle-Angriff

Die Berechnung des geheimen Schlüssels ist nicht der einzig mögliche Angriff auf das Diffie-Hellman-Protokoll. Ein wichtiges Problem besteht darin, dass Alice und Bob nicht sicher sein können, dass die Nachricht tatsächlich vom jeweils anderen kommt. Ein Angreifer Oskar kann z. B. die *Man-In-The-Middle-Attacke* verwenden. Er tauscht sowohl mit Alice als auch mit Bob einen geheimen Schlüssel aus. Alice glaubt, der Schlüssel komme von Bob und Bob glaubt, der Schlüssel komme von Alice. Alle Nachrichten, die Alice dann an Bob sendet, fängt Oskar ab, entschlüsselt sie und verschlüsselt sie mit dem zweiten Schlüssel und sendet sie an Bob.

8.6.6 Andere Gruppen

Man kann das Diffie-Hellman-Verfahren in allen zyklischen Gruppen sicher implementieren, in denen die Gruppenoperationen effizient realisierbar sind und in denen das Diffie-

Hellman-Problem bzw. das Decisional-Diffie-Hellman-Problem schwer zu lösen sind. Insbesondere muss es schwer sein, diskrete Logarithmen in G zu berechnen. In Kap. 13 werden wir Beispiele für solche Gruppen noch behandeln. Hier schildern wir nur das Prinzip.

Alice und Bob einigen sich auf eine endliche zyklische Gruppe G und einen Erzeuger g von G. Sei n die Ordnung von G. Um zu vermeiden, dass der Angriff gegen DDH aus Abschn. 8.6.3 angewendet werden kann, wählt man G so, dass n eine Primzahl ist. Alice wählt eine natürliche Zahl $a \in \{0, \ldots, n-1\}$ zufällig. Sie berechnet

$$A = g^a$$

und schickt das Ergebnis A an Bob. Bob wählt eine natürliche Zahl $b \in \{0, \ldots, n-1\}$ zufällig. Er berechnet

$$B = g^b$$

und schickt das Ergebnis an Alice. Alice berechnet nun

$$B^a = g^{ab}$$

und Bob berechnet

$$A^b = g^{ab}.$$

Der gemeinsame Schlüssel ist

$$K = g^{ab}.$$

Die Probleme CDH und DDH aus Abschn. 8.6.3 können leicht auf andere endliche Gruppen verallgemeinert werden wie auch der dort beschriebene Angriff. Er wird vermieden, wenn Gruppen von Primzahlordnung verwendet werden.

8.7 Das ElGamal-Verschlüsselungsverfahren

Das ElGamal-Verfahren hängt eng mit dem Diffie-Hellman-Schlüsselaustausch zusammen. Seine Sicherheit beruht auf der Schwierigkeit, das Diffie-Hellman-Problem in $(\mathbb{Z}/p\mathbb{Z})^*$ zu lösen.

8.7.1 Schlüsselerzeugung

Alice wählt einen Sicherheitsparameter k, eine k-Bit Primzahl p und eine Primitivwurzel $g \bmod p$ wie in Abschn. 2.22 beschrieben. Dann wählt sie zufällig und gleichverteilt einen Exponenten $a \in \{0, \ldots, p-2\}$ und berechnet

$$A = g^a \bmod p.$$

Der öffentliche Schlüssel von Alice ist (p, g, A). Der private Schlüssel von Alice ist der Exponent a. Man beachte, dass Alice im Diffie-Hellman-Verfahren den Wert A an Bob schickt. Im ElGamal-Verfahren liegt also der Schlüsselanteil von Alice ein für allemal fest und ist öffentlich.

Es wird sich in Abschn. 8.7.7 herausstellen, dass die hier beschriebene Wahl keine Chosen-Plaintext-Sicherheit des ElGamal-Verfahrens ermöglicht, weil das Decisional-Diffie-Hellman-Problem in der primen Restklassengruppe modulo p unsicher ist (siehe Abschn. 8.6.3). Wählt man $p = lq + 1$ mit einer hinreichend großen Primzahl q und ersetzt g durch $g^l \bmod p$, dann wird ElGamal in einer Untergruppe von $(\mathbb{Z}/p\mathbb{Z})^*$ der Ordnung q implementiert, in der DDH als schwer gilt.

8.7.2 Verschlüsselung

Der Klartextraum ist die Menge $\{1, \ldots, p-1\}$. Um einen Klartext m zu verschlüsseln, besorgt sich Bob den öffentlichen Schlüssel (p, g, A) von Alice. Er wählt eine Zufallszahl $b \in \{0, \ldots, p-2\}$ und berechnet

$$B = g^b \bmod p.$$

Die Zahl B ist der Schlüsselanteil von Bob aus dem Diffie-Hellman-Verfahren. Bob bestimmt

$$c = A^b m \bmod p.$$

Bob multipliziert also den Klartext m mit dem Diffie-Hellman-Schlüssel $A^b \bmod p = g^{ab} \bmod p$. Der Schlüsseltext ist (B, c).

8.7.3 Entschlüsselung

Alice hat von Bob den Schlüsseltext (B, c) erhalten, und sie kennt ihren geheimen Schlüssel a. Um den Klartext m zu rekonstruieren, bestimmt Alice den Exponenten $x = p-1-a$. Weil $0 \leq a \leq p-2$ ist, gilt $1 \leq x \leq p-1$. Dann berechnet Alice $B^x c \bmod p$. Dies ist der ursprüngliche Klartext, wie die folgende Rechnung zeigt:

$$B^x c \equiv g^{b(p-1-a)} A^b m \equiv (g^{p-1})^b (g^a)^{-b} A^b m \equiv A^{-b} A^b m \equiv m \bmod p.$$

Beispiel 8.14 Alice wählt $p = 23$, $g = 7$, $a = 6$ und berechnet den Wert $A = g^a \bmod p = 4$. Ihr öffentlicher Schlüssel ist dann $(p = 23, g = 7, A = 4)$. Ihr geheimer Schlüssel ist $a = 6$. Bob will $m = 7$ verschlüsseln. Er wählt $b = 3$, berechnet $B = g^b \bmod p = 21$ und $c = A^b m \bmod p = 11$. Der Schlüsseltext ist $(B, c) = (21, 11)$. Alice entschlüsselt $B^{p-1-6} c \bmod p = 7 = m$.

8.7.4 Effizienz

Die ElGamal-Entschlüsselung erfordert eine modulare Exponentiation wie das RSA-Verfahren. Die ElGamal-Entschlüsselung kann jedoch nicht mit dem chinesischen Restsatz beschleunigt werden.

Die Verschlüsselung mit dem ElGamal-Verfahren erfordert zwei modulare Exponentiationen, nämlich die Berechnung von $A^b \bmod p$ und $B = g^b \bmod p$. Im Gegensatz dazu erfordert die Verschlüsselung beim RSA-Verfahren nur eine Exponentiation modulo n. Die Primzahl p im ElGamal-Verfahren und der RSA-Modul n sind von derselben Größenordnung. Daher sind die einzelnen modularen Exponentiationen gleich teuer. Beim ElGamal-Verfahren kann Bob die Werte $A^b \bmod p$ und $B = g^b \bmod p$ aber auf Vorrat vorberechnen, weil sie unabhängig von der zu verschlüsselnden Nachricht sind. Die vorberechneten Werte müssen nur in einer sicheren Umgebung, etwa auf einer Chipkarte, gespeichert werden. Dann benötigt die aktuelle Verschlüsselung nur eine Multiplikation modulo p. Das ist effizienter als die RSA-Verschlüsselung.

Beispiel 8.15 Wie in Beispiel 8.14 ist der öffentliche Schlüssel von Alice ($p = 23, g = 7, A = 4$). Ihr geheimer Schlüssel ist $a = 6$. Bevor Bob in die Situation kommt, eine Nachricht wirklich verschlüsseln zu müssen, wählt er $b = 3$ und berechnet $B = g^b \bmod p = 21$ und $K = A^b \bmod p = 18$. Später will Bob $m = 7$ verschlüsseln. Dann berechnet er einfach $c = K * m \bmod 23 = 11$. Der Schlüsseltext ist $(B, c) = (21, 11)$. Wieder entschlüsselt Alice $B^{p-1-6}c \bmod p = 7 = m$.

Ein Effizienznachteil des ElGamal-Verfahrens besteht darin, dass der Schlüsseltext doppelt so lang ist wie der Klartext. Das nennt man *Nachrichtenexpansion*. Dafür ist das ElGamal-Verfahren aber ein randomisiertes Verschlüsselungsverfahren. Wir werden darauf in Abschn. 8.7.7 eingehen.

Die Länge der öffentlichen Schlüssel im ElGamal-Verfahren kann verkürzt werden, wenn im gesamten System dieselbe Primzahl p und dieselbe Primitivwurzel $g \bmod p$ verwendet wird. Dies kann aber auch ein Sicherheitsrisiko sein, wenn sich herausstellt, dass für die verwendete Primzahl die Berechnung diskreter Logarithmen einfach ist.

8.7.5 ElGamal und Diffie-Hellman

Wer diskrete Logarithmen mod p berechnen kann, ist auch in der Lage, das ElGamal-Verfahren zu brechen. Er kann nämlich aus A den geheimen Exponenten a ermitteln und dann $m = B^{p-1-a}c \bmod p$ berechnen. Es ist aber nicht bekannt, ob jemand, der das ElGamal-Verfahren brechen kann, auch diskrete Logarithmen mod p bestimmen kann.

Das ElGamal-Verfahren ist aber genauso schwer zu brechen wie das Diffie-Hellman-Verfahren. Das kann man folgendermaßen einsehen. Angenommen, Oskar kann das Diffie-Hellmann-Problem lösen, d. h. aus p, g, A und B den geheimen Schlüssel $g^{ab} \bmod$

p berechnen. Oskar möchte einen ElGamal-Schlüsseltext (B, c) entschlüsseln. Er kennt auch den entsprechenden öffentlichen Schlüssel (p, g, A). Weil er Diffie-Hellman brechen kann, kann er $K = g^{ab} \bmod p$ bestimmen und damit den Klartext $K^{-1}c \bmod p$ rekonstruieren. Sei umgekehrt angenommen, dass Oskar das ElGamal-Verfahren brechen, also aus der Kenntnis von p, g, A, B und c die Nachricht m ermitteln kann, und zwar für jede beliebige Nachricht m. Wenn er den Schlüssel g^{ab} aus p, g, A, B ermitteln will, wendet er das Entschlüsselungsverfahren für ElGamal mit $p, g, A, B, c = 1$ an und erhält eine Nachricht m. Er weiß, dass $1 = g^{ab}m \bmod p$ ist. Daher kann er $g^{ab} \equiv m^{-1} \bmod p$ berechnen und hat den Schlüssel $g^{ab} \bmod p$ gefunden.

8.7.6 Parameterwahl

Um entsprechende Sicherheit zu garantieren, muss die Primzahl gemäß Tab. 8.4 gewählt werden. Außerdem muss die Primzahl p einige Zusatzbedingungen erfüllen, die es verhindern, dass diskrete Logarithmen in $(\mathbb{Z}/p\mathbb{Z})^*$ mit bekannten Methoden (Pohlig-Hellman-Algorithmus, Number-Field-Sieve) leicht berechnet werden können. Da man aber nicht alle möglichen Algorithmen zur Lösung des Diffie-Hellman-Problems vorhersagen kann, erscheint es am sichersten, die Primzahl p zufällig und gleichverteilt zu wählen.

Bei jeder neuen ElGamal-Verschlüsselung muss Bob einen neuen Exponenten b wählen. Wählt Bob nämlich zweimal dasselbe b und berechnet damit aus den Klartexten m und m' die Schlüsseltexte

$$c = A^b m \bmod p, \quad c' = A^b m' \bmod p,$$

so gilt

$$c'c^{-1} \equiv m'm^{-1} \bmod p.$$

Jeder Angreifer, der den Klartext m kennt, kann dann den Klartext m' gemäß der Formel

$$m' = c'c^{-1}m \bmod p$$

berechnen. Damit ist also ein Known-Plaintext-Angriff möglich.

8.7.7 Chosen-Plaintext-Sicherheit

Wir zeigen in diesem Abschnitt, dass das ursprüngliche ElGamal-Verfahren nicht Chosen-Plaintext-sicher ist, aber eine kleine Modifikation diese Eigenschaft hat. Der Chosen-Plaintext-Angreifer 8.2 verwendet dieselbe Methode wie der DDH-Angreifer 8.1. Er hat Zugriff auf das Orakel $\mathbf{Enc}_{b,e}$, das ein Paar von Klartexten (m_0, m_1) erhält und die Verschlüsselung von m_b mit dem öffentlichen Schlüssel e zurückgibt.

Angreifer 8.2 ($A^{\mathbf{Enc}_{b,e}}(p, g, A)$)

(Chosen-Plaintext-Angreifer auf das ElGamal-Verfahren)

Wähle quadratischen Nichtrest m_0 modulo p

Wähle quadratischen Rest m_1 modulo p

$(B, c) \leftarrow \mathbf{Enc}_{b,e}(m_0, m_1)$

if $\left(\frac{A}{p}\right) = 1$ oder $\left(\frac{B}{p}\right) = 1$ **then**

  $l \leftarrow 1$

else

  $l \leftarrow -1$

end if

if $l\left(\frac{c}{p}\right) = 1$ **then**

  return 1

else

  return 0

end if

Theorem 8.7 *Der Vorteil des Chosen-Plaintext-Angreifers* 8.2 *ist* 1.

Beweis Aufgrund von Theorem 8.5 ist der Wert l, den der Angreifer berechnet, das Legendre-Symbol des Schlüssels K aus dem ElGamal-Verfahren. Das Orakel $\mathbf{Enc}_{b,e}$ verschlüsselt den Klartext m_b zu (c, B). Da $c \equiv Km_b \bmod p$ gilt, berechnet sich das Legendre-Symbol von m_b zu $l\left(\frac{c}{p}\right)$. Da die Legendre-Symbole von m_0 und m_1 verschieden sind, kann der Angreifer aus dem Legendre-Symbol von c erschließen, welcher Klartext verschlüsselt wurde. □

Jetzt beweisen wir, dass die Sicherheit des ElGamal-Verfahrens gegen Chosen-Plaintext-Angriffe auf das Decisional-Diffie-Hellman-Problem in der zugrundeliegenden Gruppe reduziert werden kann. Sei also A ein Angreifer auf das ElGamal-Verfahren. Algorithmus 8.2 löst das DDH-Problem unter Verwendung von A. Seine Eingabe besteht aus p, A, B, C. Er muss entscheiden, ob (A, B, C) ein Diffie-Hellman-Tripel oder ein zufälliges Tripel ist. Dazu benutzt er den den Angreifer $A^{\mathbf{Enc}_{b,e}}$ als Unterprogramm. Dieser Angreifer hat Zugriff auf das Orakel $\mathbf{Enc}_{b,e}$, das bei Eingabe eines Klartextpaares (P_0, P_1) die ElGamal-Verschlüsselung des Klartextes P_b ausgibt. Der DDH-Unterscheider implementiert ein solches Orakel und stellt es dem Angreifer zur Verfügung. Die Implementierung funktioniert so: Es gibt bei Eingabe eines Klartextpaars (m_0, m_1) den Wert $(Cm_b \bmod p, B)$ zurück, wobei C die dritte Komponente aus dem Eingabetripel des Algorithmus ist.

Algorithmus 8.2 ($\mathrm{DDH}^A(p, A, B, C)$)

(DDH-Unterscheider, der einen Chosen-Plaintext-Angreifer auf das ElGamal-Verfahren benutzt)

$b \xleftarrow{\$} \mathbb{Z}_2$
$b' \leftarrow A^{\mathbf{Enc}_{b,e}}(p, g, A)$
if $b = b'$ **then**
  return 1
else
  return 0
end if

Wir beweisen den Reduktionssatz.

Theorem 8.8 *Sei k die Bitlänge von p, sei $T(k)$ die Laufzeit des ElGamal-Angreifers A und sei $\mathbf{Adv}(k)$ sein Vorteil. Dann ist die Laufzeit von Algorithmus 8.2 $O(k) + T(k)$ und seine Erfolgswahrscheinlichkeit mindestens $1/2 + \mathbf{Adv}(k)/4$.*

Beweis Die Aussage über die Laufzeit kann aus der Konstruktion des von Algorithmus 8.2 leicht verifiziert werden.

Wir berechnen die Erfolgswahrscheinlichkeit des DDH-Algorithmus. Angenommen, der Vorteil des ElGamal-Angreifers ist $\varepsilon = \mathbf{Adv}(k)$. Wenn (A, B, C) ein Diffie-Hellman-Tripel ist, stimmt der Schlüsseltext, den $\mathbf{Enc}_{b,e}$ berechnet. Also ist die Erfolgswahrscheinlichkeit des DDH-Algorithmus in diesem Fall $(1 + \varepsilon)/2$. Falls (A, B, C) aber kein Diffie-Hellman-Tripel ist, kann der ElGamal-Angreifer im schlechtesten Fall nur raten, welcher Klartext verschlüsselt wurde. Seine Erfolgswahrscheinlichkeit ist $1/2$. Da beide Fälle mit Wahrscheinlichkeit $1/2$ auftreten, ist die Erfolgswahrscheinlichkeit des DDH-Algorithmus also

$$(1/2)(1 + \varepsilon)/2 + 1/4 = 1/2(1 + \varepsilon/2). \qquad \square$$

Theorem 8.8 zeigt, dass der Vorteil des DDH-Algorithmus mindestens halb so groß wie der Vorteil des ElGamal-Angreifers. Solange das DDH-Problem also schwierig ist, ist ElGamal sicher. Wählt man $p = kq + 1$ mit hinreichend großen Primzahlen p und q, und implementiert das ElGamal-Verfahren in der Untergruppe der Ordnung q der primen Restklassenguppe modulo p, so gilt nach heutiger Kenntnis das DDH-Problem als schwer. Theorem 8.8 impliziert die Sicherheit des ElGamal-Verfahrens gegen Chosen-Plaintext-Angriffe. Die Größe von p ergibt sich aus Tab. 8.4 und die Größe von q aus Tab. 8.5.

8.7.8 Chosen-Ciphertext-Sicherheit

Das ElGamal-Verschlüsselungsverfahren ist nicht sicher gegen Chosen-Ciphertext-Angriffe. Folgender Chosen-Ciphertext-Angriff kann nämlich jeden Schlüsseltext (B, c) entschlüsseln. Sei (p, g, A) der öffentliche ElGamal-Schlüssel. Sei außerdem o die Ordnung von g modulo p. Der Angreifer wählt einen Exponenten $r \in \{2, \ldots, o-1\}$ und einen ElGamal Klartext m_1. Dann setzt er

$$(B', c') = (g^r B, A^r c m_1) \bmod p. \tag{8.13}$$

Der Angreifer lässt sich den modifizierten Schlüsseltext (B', g') entschlüsseln. Der entsprechende Klartext sei m_2, dann ist die Entschlüsselung von (B, c) der Klartext

$$m = m_1^{-1} m_2. \tag{8.14}$$

Dass das stimmt, sieht man so. Der ursprüngliche Schlüsseltext ist

$$(B, c) = (g^b, A^b m) \bmod p \tag{8.15}$$

mit einem Exponenten $b \in \{2, \ldots o-1\}$. Darum ist

$$(B', c') = (g^{b+r}, A^{b+r} m m_1) \bmod p. \tag{8.16}$$

Die Entschlüsselung dieses Chiffretextes liefert $m_2 = m m_1 \bmod p$. Darum wird in (8.14) m richtig berechnet.

8.7.9 Homomorphie

Der Chosen-Ciphertext-Angriff aus Abschn. 8.7.8 beruht darauf, dass das ElGamal-Verschlüsselungsverfahren *homomorph* bezüglich Multiplikation ist. Um dies zu erklären, verwenden wir den öffentlichen ElGamal-Schlüssel (p, g, A). Sind dann

$$(B_i, c_i) = (g^{b_i} \bmod p, A^{b_i} m_i \bmod p), \quad i = 1, 2 \tag{8.17}$$

Verschlüsselungen der Klartexte m_1 und m_2, dann ist

$$(B_1 B_2 \bmod p, c_1 c_2 \bmod p) = (g^{b_1+b_2} \bmod p, A^{b_1+b_2} m_1 m_2 \bmod p) \tag{8.18}$$

eine ElGamal Verschlüsselung des Produktes $m_1 m_2 \bmod p$. Eine solche Eigenschaft ist zum Beispiel bei Verfahren, die elektronische Wahlen implementieren, nützlich.

8.7.10 Verallgemeinerung

Der wichtigste Vorteil des ElGamal-Verfahrens besteht darin, dass es sich nicht nur in der primen Restklassengruppe modulo einer Primzahl, sondern in jeder anderen zyklischen Gruppe G verwenden lässt. Es ist nur erforderlich, dass sich die Schlüsselerzeugung, Verschlüsselung und Entschlüsselung effizient durchführen lassen und dass das entsprechende Diffie-Hellman-Problem schwer zu lösen ist. Insbesondere muss die Berechnung diskreter Logarithmen in G schwer sein, weil sonst das Diffie-Hellman-Problem leicht zu lösen ist.

Es ist sehr wichtig, dass das ElGamal-Verfahren verallgemeinert werden kann, weil es immer möglich ist, dass jemand einen effizienten Algorithmus zur Berechnung diskreter Logarithmen in $(\mathbb{Z}/p\mathbb{Z})^*$ findet. Dann kann man das ElGamal-Verfahren in $(\mathbb{Z}/p\mathbb{Z})^*$ nicht mehr benutzen, weil es unsicher geworden ist. In anderen Gruppen kann das ElGamal-Verfahren aber immer noch sicher sein, und man kann dann diese Gruppen verwenden.

Folgende Gruppen sind z. B. zur Implementierung des ElGamal-Verfahrens geeignet:

1. Die Punktgruppe einer elliptischen Kurve über einem endlichen Körper.
2. Die Jakobische Varietät hyperelliptischer Kurven über endlichen Körpern.
3. Die Klassengruppe imaginär-quadratischer Ordnungen.

8.8 Übungen

Übung 8.1 Man zeige, dass man im RSA-Verfahren den Entschlüsselungsexponenten d auch so wählen kann, dass $de \equiv 1 \bmod \operatorname{lcm}(p-1, q-1)$ ist.

Übung 8.2 Bestimmen Sie alle für den RSA-Modul $n = 437$ möglichen Verschlüsselungsexponenten. Geben Sie eine Formel für die Anzahl der für einen RSA-Modul n möglichen Verschlüsselungsexponenten an.

Übung 8.3 Erzeugen Sie zwei 8-Bit-Primzahlen p und q so, dass $n = pq$ eine 16-Bit-Zahl ist und der öffentliche RSA-Schlüssel $e = 5$ verwendet werden kann. Berechnen Sie den privaten Schlüssel d zum öffentlichen Schlüssel $e = 5$. Verschlüsseln Sie den String 110100110110111 mit dem öffentlichen Exponenten 5.

Übung 8.4 Alice verschlüsselt die Nachricht m mit Bobs öffentlichem RSA-Schlüssel $(899, 11)$. Der Schlüsseltext ist 468. Bestimmen Sie den Klartext.

Übung 8.5 Entwerfen Sie einen polynomiellen Algorithmus, der bei Eingabe von natürlichen Zahlen c und e entscheidet, ob c eine e-te Potenz ist und wenn dies der Fall ist, auch noch die e-te Wurzel von c berechnet. Beweisen Sie, dass der Algorithmus tatsächlich polynomielle Laufzeit hat.

Übung 8.6 Implementieren Sie den Algorithmus aus Übung 8.5.

Übung 8.7 Wieviele Operationen erfordert die RSA-Verschlüsselung mit Verschlüsselungsexponent $e = 2^{16} + 1$?

Übung 8.8 Die Nachricht m wird mit den öffentlichen RSA-Schlüsseln $(391, 3)$, $(55, 3)$ und $(87, 3)$ verschlüsselt. Die Schlüsseltexte sind 208, 38 und 32. Verwenden Sie die Low-Exponent-Attacke, um m zu finden.

Übung 8.9 (Common-Modulus-Attacke) Wenn man mit dem RSA-Verfahren eine Nachricht m zweimal verschlüsselt, und zwar mit den öffentlichen Schlüsseln (n, e) und (n, f), und wenn $\gcd(e, f) = 1$ gilt, dann kann man den Klartext m aus den beiden Schlüsseltexten $c_e = m^e \bmod n$ und $c_f = m^f \bmod n$ berechnen. Wie geht das?

Übung 8.10 Die Nachricht m wird mit den öffentlichen RSA-Schlüsseln $(493, 3)$ und $(493, 5)$ verschlüsselt. Die Schlüsseltexte sind 293 und 421. Verwenden Sie die Common-Modulus-Attacke, um m zu finden.

Übung 8.11 Sei $n = 1591$. Der öffentliche RSA-Schlüssel von Alice ist (n, e) mit minimalem e. Alice erhält die verschlüsselte Nachricht $c = 1292$. Entschlüsseln Sie diese Nachricht mit Hilfe des chinesischen Restsatzes.

Übung 8.12 Angenommen, der RSA-Modul ist $n = 493$, der Verschlüsselungsexponent ist $e = 11$, und der Entschlüsselungsexponent ist $d = 163$. Verwenden Sie die Methode aus Abschn. 8.3.4, um n zu faktorisieren.

Übung 8.13 (Cycling-Attacke) Sei (n, e) ein öffentlicher RSA-Schlüssel. Für einen Klartext $m \in \{0, 1, \ldots, n-1\}$ sei $c = m^e \bmod n$ der zugehörige Schlüsseltext. Zeigen Sie, dass es eine natürliche Zahl k gibt mit

$$m^{e^k} \equiv m \bmod n.$$

Beweisen Sie für ein solches k:

$$c^{e^{k-1}} \equiv m \bmod n.$$

Ist dies eine Bedrohung für RSA?

Übung 8.14 Sei $n = 493$ und $e = 3$. Bestimmen Sie den kleinsten Wert von k, für den die Cycling-Attacke aus Übung 8.13 funktioniert.

Übung 8.15 Bob verschlüsselt Nachrichten an Alice mit dem Rabin-Verfahren. Er verwendet dieselben Parameter wie in Beispiel 8.9. Die Klartexte sind Blöcke in $\{0,1\}^8$, in denen die ersten beiden Bits mit den letzten beiden Bits übereinstimmen. Kann Alice alle Klartexte eindeutig entschlüsseln?

Übung 8.16 Sei $n = 713$ ein öffentlicher Rabin-Schlüssel und sei $c = 289$ ein Schlüsseltext, den man durch Rabin-Verschlüsselung mit diesem Modul erhalten hat. Bestimmen Sie alle möglichen Klartexte.

Übung 8.17 Übertragen Sie die Low-Exponent-Attacke und die Multiplikativitäts-Attacke, die beim RSA-Verfahren besprochen wurden, auf das Rabin-Verfahren und schlagen Sie entsprechende Gegenmaßnahmen vor.

Übung 8.18 Wie kann der Rabin-Modul $n = 713$ mit zwei möglichen Klartexten aus Übung 8.16 faktorisiert werden?

Übung 8.19 Wie kann man aus zwei ElGamal-Schlüsseltexten einen dritten ElGamal-Schlüsseltext machen, ohne den geheimen ElGamal-Schlüssel zu kennen? Wie kann man diesen Angriff verhindern?

Übung 8.20 Alice erhält den ElGamal-Chiffretext $(B = 30, c = 7)$. Ihr öffentlicher Schlüssel ist $(p = 43, g = 3)$. Bestimmen Sie den zugehörigen Klartext.

Übung 8.21 Der öffentliche ElGamal-Schlüssel von Bob sei $p = 53$, $g = 2$, $A = 30$. Alice erzeugt damit den Schlüsseltext $(24, 37)$. Wie lautet der Klartext?