Digitale Signaturen

12

12.1 Idee

Eine weitere wichtige kryptographische Komponente ist die *elektronische* oder *digitale Signatur*. Ein elektronisches oder digitales *Signaturverfahren* besteht aus einem *Schlüsselerzeugungsalgorithmus*, einem *Signieralgorithmus* und einem *Verifikationsalgorithmus*. Der Schlüsselerzeugungsalgorithmus erzeugt Schlüsselpaare. In einem solchen Paar (d, e) ist d der *private Signierschlüssel* und e der zugehörige *öffentliche Verifikationsschlüssel*. Der Signieralgorithmus berechnet aus einem Dokument oder Datensatz x und einem Signierschlüssel d eine *elektronische* oder *digitale Signatur* s. Der Verifikationsalgorithmus bekommt als Eingabe das Dokument x, eine Signatur s und einen öffentlichen Schlüssel e. Wurde s mit dem zu e gehörenden Signierschlüssel d aus x berechnet, so gibt der Verfifiktionsalgorithms „gültig“ zurück und anderfalls „ungültig“.

Was beweist eine gültige digitale Signatur s eines Dokumentes d? Erstens zeigt sie die *Integrität* des Dokumentes: Es wurde seit der Erstellung der Signatur nicht geändert. Zweitens beweist die Signatur die *Authentizität* von d: Die Inhaberin Alice des geheimen Signierschlüssels ist die Urheberin des Dokumentes. Wenn x ein Vertrag ist, kann das zum Beispiel bedeuten, dass Alice dem Vertrag zustimmt. Drittens sorgt die Signatur für *Nicht-Abstreitbarkeit*: Alice kann auch zu einem späteren Zeitpunkt nicht bestreiten, dass sie die Urheberin des Dokumentes d ist, also zum Beispiel einen Vertrag unterschrieben hat. Digitale Signaturen können nämlich zu jedem Zeitpunkt allen verifiziert werden. Man nennt das *universelle Verifizierbarkeit*.

Die Anwendungen von elektronischen Signaturen sind vielfältig. Eine wichtige Anwendung ist der Authentizitätsnachweis für Software. So werden zum Beispiel täglich Milliarden von Updates für Anti-Virus-Software auf Computern installiert. Dabei ist es wichtig, dass sich die Computer, auf dem die Updates installiert werden, von ihrer Authentizität überzeugen können, also davon, dass sie wirklich vom Hersteller der Software kommt und es sich nicht um Schadsoftware von Dritten handelt. Installiert der Computer nämlich statt des Updates solche Schadsoftware, kann das verheerende Folgen haben.

J. Buchmann, *Einführung in die Kryptographie*, Springer-Lehrbuch,
DOI 10.1007/978-3-642-39775-2_12

Die Schadsoftware kann zum Beispiel den Computer ausspionieren oder die Festplatte löschen. Andere Anwendungen elektronischer Signaturen sind zum Beispiel die Authentisierung von Emails und die Authentisierung und Nicht-Abstreitbarkeit elektronischer Steuererklärungen.

Im nächsten Abschnitt werden digitale Signaturen formal definiert.

12.2 Definition

Die Definition von digitalen Signaturverfahren ähnelt der Definition von Public-Key-Verschlüsselungsverfahren.

Definition 12.1 Ein *digitales* oder *elektronisches Signaturverfahren* ist ein Tupel $(\mathbf{K}, \mathbf{M}, \mathbf{S}, \mathbf{KeyGen}, \mathbf{Sign}, \mathbf{Ver})$ mit folgenden Eigenschaften:

1. $\mathbf{K}$ ist eine Menge von Paaren. Sie heißt *Schlüsselraum*. Ihre Elemente heißen *Schlüsselpaare*. Ist $(e, d) \in \mathbf{K}$, so heißt die Komponente d *Signierschlüssel* oder *privater Schlüssel* und die Komponente e *Verifikationsschlüssel* oder *öffentlicher Schlüssel*.
2. $\mathbf{M}$ ist eine Menge. Sie heißt *Nachrichtenraum*. Ihre Elemente heißen *Nachrichten*.
3. $\mathbf{S}$ ist eine Menge. Sie heißt *Signaturraum*. Ihre Elemente heißen *Signaturen*.
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 $\mathbf{M}(d) \subset \mathbf{P}$ die Menge der Nachrichten, die mit einem privaten Schlüssel d signiert werden können.
5. **Sign** ist ein probabilistischer Polynomzeit-Algorithmus, der zustandsbehaftet sein kann. Er heißt *Signieralgorithmus*. Bei Eingabe von 1^k, $k \in \mathbb{N}$, eines privaten Schlüssels $d \in \mathbf{Priv}(k)$ und einer Nachricht $m \in \mathbf{M}(d)$ gibt er eine Signatur $s \in \mathbf{S}$ zurück.
6. **Ver** ist ein deterministischer Algorithmus. Er heißt *Verifikationsalgorithmus*. Er gibt bei Eingabe von 1^k, $k \in \mathbb{N}$, eines öffentlichen Schlüssels $e \in \mathbf{Pub}(k)$, einer Nachricht $m \in \mathbf{M}$ und einer Signatur $s \in \mathbf{S}$ ein Bit $b \in \{0, 1\}$ aus. Ist $b = 1$, so heißt die Signatur *gültig* und andernfalls *ungültig*.
7. Das Signaturverfahren veirfiziert korrekt: für alle $k \in \mathbb{N}$, jedes Schlüsselpaar $(e, d) \in \mathbf{K}(k)$, jede Nachricht $m \in \mathbf{M}(d)$ und jede Signatur $s \in \mathbf{S}$ gilt $\mathbf{Ver}(1^k, e, m, s) = 1$ genau dann, wenn s ein Rückgabewert von $\mathbf{Sign}(1^k, d, m)$ ist.

Die meisten Signaturverfahren können beliebig lange Bitstrings signieren. Es ist dann $\mathbf{M}(d) = \mathbb{Z}_2^*$ für alle privaten Schlüssel d. Es gibt aber auch andere Fälle, wie wir zum Beispiel in Abschn. 12.3 zeigen werden.

12.3 Das Lamport-Diffie-Einmal-Signaturverfahren

In diesem Abschnitt behandeln wir ein besonders einfaches Signaturverfahren: das *Lamport-Diffie-Einmal-Signaturverfahren* [41]. Wir bezeichnen es im Folgenden kurz mit LD-OTS, wobei OTS für *One-Time Signature Scheme* steht. Der Name kommt daher, dass in LD-OTS jedes Schlüsselpaar nur einmal benutzt werden darf. Im Folgenden sei k ein fest gewählter Sicherheitsparameter.

Für die Konstruktion der Schlüssel, die Signatur und die Verifikation verwendet LD-OTS eine Einwegfunktion

$$h : \{0,1\}^k \to \{0,1\}^k, k \in \mathbb{N}. \tag{12.1}$$

Einzige Sicherheitsvoraussetzung von LD-OTS ist die Einwegeigenschaft von h. Dies werden wir in Abschn. 12.10.4 begründen. Diese Voraussetzung ist minimal, weil die Existenz von sicheren Signaturverfahren und die Existenz von Einwegfunktionen äquivalent ist (siehe [60]). LD-OTS ist sehr flexibel. Jede neue Einwegfunktion führt zu einer neuen Variante von LD-OTS. Kandidaten für Einwegfunktionen können zum Beispiel mit Hilfe von kryptographischen Hashfunktionen und Blockchiffren konstruiert werden. Es ist aber keine Funktion h bekannt, die nachweislich die Einwegeigenschaft hat. Darum besteht immer die Möglichkeit, dass sich vermeintliche Einwegfunktionen als leicht invertierbar herausstellen. Wegen der großen Flexibilität von LD-OTS kann die unsichere Funktion dann leicht durch einen neuen Kandidaten ersetzt werde.

12.3.1 Schlüsselerzeugung

Die Signier- und Verifikationsschlüssel bestehen aus $2k$ Bitstrings der Länge k, also ist $\mathbf{Priv}(k) = \mathbf{Pub}(k) = \mathbb{Z}_2^{(k,2k)}$. Ein LD-OTS-Signierschlüssel ist also eine Matrix

$$x = \big(x(0,1), x(1,1), x(0,2), x(1,2), \ldots, x(0,k), x(1,k)\big) \in \mathbb{Z}_2^{(k,2k)}.$$

Der zugehörige Verifikationsschlüssel ensteht, indem die Einwegfunktion auf Spalten des Signierschlüssels h angewendet wird. Er ist also

$$\begin{aligned} y &= \big(y(0,1), y(1,1), y(0,2), y(1,2), \ldots, y(0,k), y(1,k)\big) \\ &= \big(h(x(0,1)), h(x(1,1)), h(x(0,2)), h(x(1,2)), \ldots, h(x(0,k), h(x(1,k))\big). \end{aligned}$$

Beispiel 12.1 Wir nehmen an, dass der Sicherheitsparameter $k = 3$ ist. Die Funktion h bildet (x_1, x_2, x_3) auf (x_3, x_2, x_1) ab. Es ist also

$$h(011) = 110, \quad h(001) = 100.$$

Das ist zwar keine Einwegfunktion, weil man die Urbilder leicht durch Ausprobieren findet, aber für $k = 3$ gibt es ohnehin keine Einwegfunktionen. Das Beispiel dient nur der Illustration. Der Signierschlüssel sei

$$(x(0,1), x(1,1), x(0,2), x(1,2), x(0,3), x(1,3)) = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}.$$

Der Verifikationsschlüssel ist dann

$$(y(0,1), y(1,1), y(0,2), y(1,2), y(0,3), y(1,3)) = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}.$$

12.3.2 Signatur

Bei Sicherheitsparameter k signiert LD-OTS Bitstrings der Länge k. Es ist also $\mathbf{M}(d) = \mathbb{Z}_2^k$ für alle Signierschlüssel $d \in \mathbf{Priv}(k)$. Sollen beliebig lange Dokumente signiert werden, kann zusätzlich eine kryptographische Hashfunktion verwendet werden, die solche Dokumente auf Strings der Länge k abbildet. Dann ist $\mathbf{M}(d) = \mathbb{Z}_2^*$ für $d \in \mathbf{Priv}(k)$.

Die Signatur einer Nachricht $m = (m_1, \ldots, m_k) \in \{0,1\}^k$ ist die Matrix

$$s = (s_1, \ldots, s_k) = (x(m_1, 1)), \ldots, x(m_k, k)).$$

Die Spalten der Signatur sind Spalten des Signierschlüssels. Ist $i \in \{1, \ldots, k\}$ und ist $m_i = 0$, so ist $x(0,i)$ die i-te Spalte in der Signatur. Ist $m_i = 1$, so ist diese Spalte $x(1,i)$. Die Signatur besteht also aus der Hälfte der Spalten des geheimen Signierschlüssels. Das ist auch der Grund dafür, dass jeder Signierschlüssel nur einmal verwendet werden darf.

Beispiel 12.2 Wir setzen Beispiel 12.1 fort. Der Signierschlüssel ist

$$(x(0,1), x(1,1), x(0,2), x(1,2), x(0,3), x(1,3)) = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}.$$

Signiert wird die Nachricht $m = (001)$. Die Signatur ist dann

$$\begin{aligned} s &= (s_1, s_2, s_3) = (x(m_1, 1), x(m_2, 2), x(m_3, 3)) \\ &= (x(0,1), x(0,2), x(1,3)) = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}. \end{aligned}$$

12.3.3 Verifikation

Der Verifizierer kennt die Einwegfunktion h, den Verifikationsschlüssel y, die Nachricht $m = (m_1, \ldots, m_k)$ und die Signatur $s = (s_1, \ldots, s_k)$. Er akzeptiert die Signatur, wenn

$$(h(s_1), \ldots, h(s_k)) = (y(m_1, 1), \ldots, y(m_k, k))$$

und weist sie andernfalls zurück.

Beispiel 12.3 Wir setzen Beispiel 12.2 fort. Der Verfizierer weiß, dass die Funktion h den String (x_1, x_2, x_3) auf (x_3, x_2, x_1) abbildet. Die signierte Nachricht ist $m = (001)$. Er kennt den Verifikationsschlüssel

$$(y(0,1), y(1,1), y(0,2), y(1,2), y(0,3), y(1,3)) = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

und die Signatur

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}.$$

Um diese Signatur zu verifizieren, berechnet der Verifizierer

$$(h(s_1), h(s_2), h(s_3)) = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

und prüft, ob

$$\begin{aligned} \big(h(s_1), h(s_2), h(s_3)\big) &= \big(y(m_1, 1), y(m_2, 2), y(m_3, 3)\big) \\ &= \big(y(0, 1), y(0, 2), y(1, 3)\big) \end{aligned} \tag{12.2}$$

gilt. Dies ist tatsächlich der Fall. Also akzeptiert der Verifizierer die Signatur.

12.4 Sicherheit

In Abschn. 8.2.1 wurde die Sicherheit von Public-Key-Verschlüsselungsverfahren diskutiert. Wir behandeln die Sicherheit von Algorithmen für digitale Signaturen analog.

12.4.1 Angriffsziele

Angreifer auf Signaturverfahren haben das Ziel, gültige Signaturen zu erzeugen ohne den entsprechenden Signierschlüssel zu kennen. Können sie diesen Signierschlüssel berechnen, ist das Verfahren *vollständig gebrochen*. Sie können dann nämlich alle Dokumente

ihrer Wahl signieren. Es ist aber auch möglich, dass Angreifer Signaturen von Dokumenten ihrer Wahl zu fälschen versuchen, ohne vorher den Signierschlüssel zu bestimmen. Dies wird *selektive Fälschung* genannt. Dabei wird die Nachricht ausgesucht und dann die passende Signatur erzeugt. Schließlich ist es ein mögliches Angriffsziel, ohne Kenntnis des Signaturschlüssels ein Paar (m, s) so zu erzeugen, dass s eine gültige Signatur der Nachricht m ist. Im Gegensatz zur selektiven Fälschung wird nicht verlangt, dass die Angreifer die Nachricht zuerst auswählen. Sie können m und s simultan berechnen. Dies nennt man *existentielle Fälschung*. Sind existentielle Fälschungen relevant? Eine Anwendung von Signaturverfahren ist Identifikation. Alice kann Bob ihre Identität zum Beispiel dadurch beweisen, dass sie Bob eine gültig signierte Nachricht schickt. Dabei kommt es nicht auf den Inhalt der Nachricht an, sondern nur darauf, dass die Signatur gültig ist. Angenommen, ein Angreifer kann eine Nachricht m und eine gültige Alice-Signatur s für m erzeugen. Dann kann er das Paar (m, s) benutzen, um Bob davon zu überzeugen, dass er Alice ist. Diese Methode der Identifikation wird normalerweise nicht verwendet. Stattdessen schickt Bob eine Nachricht an Alice, die sie signieren soll. Aber die Überlegung zeigt trotzdem, dass existentielle Fälschungen nicht harmlos sind.

Wir geben nun Beispiele, wie diese Angriffsziele erreicht werden können.

Beispiel 12.4 LD-OTS ist eigentlich ein Einmal-Signaturverfahren. Also darf jeder Signierschlüssel nur für eine Signatur verwendet werden. Aber angenommen, das wird nicht beachtet und derselbe private Schlüssel wird verwendet, um die Dokumente 0^k und 1^k zu signieren, wobei k der verwendete Sicherheitsparameter ist. Dann können Angreifer aus diesen beiden Signaturen den Signierschlüssel konstruieren. Die Signatur von 0^k enthält nämlich k Bitstrings aus dem Signierschlüssel. Die Signatur von 1^k enthält die restlichen k Bitstrings aus dem Signierschlüssel. Damit ist LD-OTS vollständig gebrochen.

Beispiel 12.5 Angenommen, der Sicherheitsparameter für LD-OTS ist gerade, also $k = 2l$, $l \in \mathbb{N}$, und in LD-OTS wird der Signierschlüssel – regelwidrig – verwendet, um die Dokumente $0^l 1^l$ und 1^{2l} zu signieren. Dann können Angreifer alle Dokumente von der Form $m_1 1^l$ signieren mit $m_1 \in \mathbb{Z}_2^l$. Die beiden Signaturen enthalten nämlich die ersten l Spalten des Signierschlüssels. Angreifer haben also die Möglichkeit einer (eingeschränkten) selektiven Fälschung.

12.4.2 Angriffstypen

Genau wie bei Verschlüsselungsverfahren können auch Angreifer von Signaturverfahren über unterschiedliche Fähigkeiten verfügen. Dies erläutern wir im Folgenden. Das Ziel der Angreifer ist eine existentielle Fälschung.

No-Message-Angriff

Bei einem *No-Message-Angriff* kennt der Angreifer nur den öffentlichen Verifikationsschlüssel und sonst nichts. Wie diese Bezeichnung zustande kommt, wird erst deutlich,

wenn die nächsten Angriffstypen besprochen werden. Dort stehen dem Angreifer nämlich Signaturen von Nachrichten (Messages) zur Verfügung.

Known-Message-Angriff

Angreifer kennen oft mehr als nur den öffentlichen Verifikationsschlüssel. Sie können zum Beispiel Kommunikationsverbindungen abhören und dabei Nachrichten und ihre Signaturen aufzeichnen. Diese Kenntnis können sie nutzen, um Signaturen zu fälschen. Dies nennt man *Known-Message-Angriff*. Beispiel 12.5 zeigt einen solchen Angriff.

Chosen-Message-Angriff

Der Angreifer kennt den Verifikationsschlüssel von Alice. Der Angreifer hat aber bei der Konstruktion einer existentiellen Fälschung mehr Möglichkeiten als der No-Message-Angreifer. Während der Konstruktion von (m, s) kann er jederzeit Alice-Signaturen von Nachrichten seiner Wahl bekommen. Diese Nachrichten dürfen nur nicht m sein. Beispiel 12.6 zeigt, dass Chosen-Message-Angriffe realistisch sind.

Beispiel 12.6 Ein Webserver will nur legitimierten Nutzern Zugang gewähren. Meldet sich ein Nutzer an, generiert der Webserver eine Zufallszahl und fordert den Nutzer auf, diese Zufallszahl zu signieren. Ist die Signatur gültig, wird der Zugang gewährt. Andernfalls nicht. Ein Angreifer kann sich als Webserver ausgeben und sich selbst gewählte Dokumente signieren lassen.

Chosen-Message-Angriffe werden in *adaptive* und *nicht-adaptive* unterschieden. Bei adaptiven Angriffen darf der Angreifer die Nachrichten, die er signieren lässt, in Abhängigkeit von den Signaturen wählen, die er vorher gesehen hat. Bei nicht-adaptiven Angriffen darf er das nicht sondern muss vor seiner Berechnung alle Signaturen anfordern. Im Folgenden verwenden wir aber den Begriff *Chosen-Message-Angriff* synonym mit *adaptiver Chosen-Message-Angriff*, weil nicht-adaptive Angriffe kaum eine Rolle spielen.

12.5 RSA-Signaturen

In Abschn. 8.3 wurde das erste Public-Key-Verschlüsselungsverfahren beschrieben: das RSA-Verfahren. Dieses Verfahren kann man auch zur Erzeugung digitaler Signaturen verwenden. Die Idee ist ganz einfach. Alice signiert ein Dokument m, indem sie das Dokument mit ihrem privaten Schlüssel d „entschlüsselt“, also die Signatur $s = m^d \bmod n$ berechnet, wobei n der RSA-Modul ist. Bob verifiziert die Signatur s, indem er sie mit seinem öffentlichen Schlüssel „verschlüsselt“, also $m' = s^e \bmod n$ berechnet. Ist m' identisch mit dem signierten Dokument m, so ist die Signatur gültig und andernfalls nicht. Es ist nämlich nach heutiger Kenntnis für hinreichend große Moduli n unmöglich, eine e-te Wurzel s modulo n aus m ohne Kenntnis von Alices privatem Schlüssel zu berechnen. Ein solches s kann also nur Alice berechnen.

12.5.1 Schlüsselerzeugung

Die Schlüsselerzeugung funktioniert genauso wie beim RSA-Verschlüsselungsverfahren, das in Abschn. 8.3.1 beschrieben wird.

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

$$1 < e < \varphi(n) = (p-1)(q-1) \text{ und } \gcd(e,(p-1)(q-1)) = 1 \tag{12.3}$$

und berechnet eine natürliche Zahl d mit

$$1 < d < (p-1)(q-1) \text{ und } de \equiv 1 \bmod (p-1)(q-1). \tag{12.4}$$

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.

12.5.2 Signatur

Wir erläutern die einfachste Variante des RSA-Signieralgorithmus. Wir werden in Abschn. 12.5.4 zeigen, dass sie existentielle Fälschungen ermöglicht. Darum beschreibt Abschn. 12.5.6 einen modifizierten RSA-Signieralgorithmus.

Signiert werden Zahlen in der Menge $\mathbb{Z}_n$. Die *Signatur* von $m \in \mathbb{Z}_n$ ist

$$s = m^d \bmod n. \tag{12.5}$$

Hierbei ist d der private Signierschlüssel.

12.5.3 Verifikation

Der Verifikationsalgorithmus erhält den öffentlichen Schlüssel (n,e), ein Dokument $m \in \mathbb{Z}_n$ und die Signatur $s \in \mathbb{Z}_n$. Um die Signatur zu verifizieren, überprüft der Algorithmus, ob

$$m \equiv s^e \bmod n \tag{12.6}$$

gilt. Dass (12.6) stimmt, wenn die Signatur s korrekt gebildet wurde, folgt aus Theorem 8.1.

Wurde die RSA-Signatur gemäß (12.5) gebildet, kann der Verifikationsalgorithmus sogar auf die Eingabe m verzichten. Die Nachricht m wird ja bei der Verifikation in (12.6) konstruiert. Wenn die Nachricht m für den Verifizierer Sinn macht, kann er sicher sein, dass er von der Besitzerin des privaten Signierschlüssels signiert wurde. Dies nennt man *Signatur mit Nachrichten-Gewinnung*. Diese Eigenschaft ermöglicht gleichzeitig existentielle Fälschungen, wie wir in Abschn. 12.5.4 zeigen werden.

Beispiel 12.7 Alice wählt $p = 11$, $q = 23$, $e = 3$. Daraus ergibt sich $n = 253$ und $d = 147$. Alices öffentlicher Verifikationsschlüssel ist $(253, 3)$. Ihr privater Signierschlüssel ist 147.

Alice will an einem Geldautomaten 111 Euro abheben und dazu diesen Betrag signieren. Ihre Chipkarte berechnet $s = 111^{147} \bmod 253 = 89$. Der Geldautomat erhält die Signatur $s = 89$. Er berechnet $m = s^3 \bmod 253 = 111$. Damit weiß der Geldautomat, dass Alice 111 Euro abheben möchte.

12.5.4 Angriffe

So, wie die Erzeugung von RSA-Signaturen bisher beschrieben wurde, bestehen eine Reihe von Gefahren.

Eine einfache existentielle Fälschung funktioniert so: Oskar wählt eine Zahl $s \in \{0, \ldots, n-1\}$. Dann behauptet er, s sei eine RSA-Signatur von Alice. Wer diese Signatur verifizieren will, berechnet $m = s^e \bmod n$ und glaubt, Alice habe m signiert. Dies ist ein No-Message-Angriff. Das nächste Beispiel zeigt, dass eine solche Fälschung gravierende Folgen haben kann.

Beispiel 12.8 Wie in Beispiel 12.7 wählt Alice $p = 11$, $q = 23$, $e = 3$. Daraus ergibt sich $n = 253$, $d = 147$. Alices öffentlicher Schlüssel ist $(253, 3)$. Ihr privater Schlüssel ist 147.

Oskar möchte von Alices Konto Geld abheben. Er verwendet eine Chipkarte, die $s = 123$ an den Geldautomaten schickt. Der Geldautomat berechnet $m = 123^3 \bmod 253 = 52$. Er glaubt, dass Alice 52 Euro abheben will. Tatsächlich hat Alice die 52 Euro aber nie unterschrieben. Oskar hat ja nur eine zufällige Unterschrift gewählt.

Eine weitere Gefahr beim Signieren mit RSA kommt daher, dass das RSA-Verfahren multiplikativ ist. Sind $m_1, m_2 \in \mathbb{Z}_n$ und sind $s_1 = m_1^d \bmod n$ und $s_2 = m_2^d \bmod n$ die Signaturen von m_1 und m_2, dann ist

$$s = s_1 s_2 \bmod n = (m_1 m_2)^d \bmod n$$

die Signatur von $m = m_1 m_2$. Chosen-Message-Angreifer können die Multiplikativität von RSA ausnutzen, um die Signatur jeder Nachricht in $\mathbb{Z}_n$ zu fälschen. Soll nämlich die Nachricht $m \in \mathbb{Z}_n$ signiert werden, so wählt der Angreifer eine andere Nachricht $m_1 \in \mathbb{Z}_n$

mit $\gcd(m_1, n) = 1$. Dann berechnet er

$$m_2 = m m_1^{-1} \bmod n.$$

Dabei ist m_1^{-1} das Inverse von m_1 mod n. Der Chosen-Message-Angreifer lässt sich die beiden Nachrichten m_1 und m_2 signieren. Er erhält die Signaturen s_1 und s_2 und kann daraus die Signatur $s = s_1 s_2 \bmod n$ von m berechnen.

Das RSA-Signaturverfahren, so wie es bis jetzt beschrieben wurde, erlaubt also existentielle Fälschungen mit Hilfe von No-Message-Angriffen und selektive Fälschungen mittels Chosen-Message-Angriffen.

In den Abschn. 12.5.5 und 12.5.6 beschreiben wir Vorkehrungen gegen die beiden beschriebenen Gefahren.

Wir weisen noch auf eine weitere Angriffsmöglichkeit hin. Zu Beginn der Verifikation einer von Alice signierten Nachricht besorgt sich Bob ihren öffentlichen Schlüssel (n, e). Wenn es dem Angreifer Oskar gelingt, Bob seinen eigenen öffentlichen Schlüssel als den Schlüssel von Alice unterzuschieben, kann er danach Signaturen erzeugen, die Bob als Signaturen von Alice anerkennt. Es ist also wichtig, dass Bob den authentischen öffentlichen Schlüssel von Alice hat. Er muss sich von der Authentizität des öffentlichen Schlüssels von Alice überzeugen können. Dazu werden Public-Key-Infrastrukturen verwendet, die in Kap. 16 diskutiert werden.

12.5.5 Signatur von Nachrichten mit Redundanz

Zwei der im vorigen Abschnitt beschriebenen Angriffe werden unmöglich, wenn nur Nachrichten $m \in \{0, 1, \ldots, n-1\}$ signiert werden, deren Binärdarstellung von der Form $w \circ w$ ist mit $w \in \{0, 1\}^*$. Die Binärdarstellung besteht also aus zwei gleichen Hälften. Der wirklich signierte Text ist die erste Hälfte, nämlich w. Aber signiert wird $w \circ w$. Bei der Verifikation wird $m = s^e \bmod n$ bestimmt, und dann wird geprüft, ob der signierte Text die Form $w \circ w$ hat. Wenn nicht, wird die Signatur zurückgewiesen.

Werden nur Nachrichten von der Form $w \circ w$ signiert, kann der Angreifer Oskar die beschriebene existentielle Fälschung nicht mehr anwenden. Er müsste nämlich eine Signatur $s \in \{0, 1, \ldots, n-1\}$ auswählen, für die die Binärentwicklung von $m = s^e \bmod n$ die Form $w \circ w$ hat. Es ist unbekannt, wie man eine solche Signatur s ohne Kenntnis des privaten Schlüssels bestimmen kann.

Auch die Multiplikativität von RSA kann nicht mehr ausgenutzt werden. Es ist nämlich äußerst unwahrscheinlich, dass $m = m_1 m_2 \bmod n$ eine Binärentwicklung von der Form $w \circ w$ hat, wenn das für beide Faktoren der Fall ist.

Die Funktion

$$R : \mathbb{Z}_2^* \to \mathbb{Z}_2^*, w \mapsto R(w) = w \circ w,$$

die zur Erzeugung der speziellen Struktur verwendet wurde, heißt *Redundanzfunktion*. Es können natürlich auch andere Redundanzfunktionen verwendet werden.

Der nächste Abschnitt erläutert eine andere Methode, die beiden ersten Angriffsmöglichkeiten aus Abschn. 12.5.4 zu verhindern. Sie hat den Vorteil, dass damit beliebig lange Nachrichten signiert werden können. Das geht mit der Redundanz-Methode nicht. Gleichzeitig hat sie den Nachteil, dass bei der Verifikation die signierte Nachricht nicht mehr rekonstruiert werden kann. Dies ist aber bei der Verwendung von Redundanz möglich.

12.5.6 Signatur mit Hashwert

In diesem Abschnitt erläutern wir eine andere Methode, dem No-Message-Angriff und dem Chosen-Message-Angriff aus Abschn. 12.5.4 durch Verwendung einer Hashfunktion zu begegnen. Diese Methode ermöglicht es gleichzeitig, beliebig lange Nachrichten zu signieren. Wir verwenden dazu eine öffentlich bekannte, kollisionsresistente Hashfunktion

$$h : \mathbb{Z}_2^* \to \mathbb{Z}_n.$$

Ausserdem benutzen wir das RSA-Schlüsselpaar $((e, n), d)$.

Der Signieralgorithmus wird folgendermaßen modifiziert. Soll $m \in \mathbb{Z}_2^*$ signiert werden, benutzt der Algorithmus die Hashfunktion h um die Signatur

$$s = h(m)^d \bmod n$$

zu berechnen. Die Signatur ist also die ursprüngliche RSA-Signatur des Hashwertes $h(m)$.

Der Verifikationsalgorithmus kennt die Hashfunktion h ebenfalls. Um die Signatur s von m zu verifizieren, berechnet er den Hashwert $h(m)$ und prüft ob

$$h(m) = s^e \bmod n$$

ist. Stimmt das, ist die Signatur gültig. Andernfalls ist sie ungültig. Diese Verifikation kann die Nachricht m nicht aus der Signatur gewinnen. Sie kann zwar den Hashwert $h(m) = s^e \bmod n$ rekonstruieren. Aber daraus kann sie nicht die Nachricht m berechnen. Das liegt daran, dass wie in Kap. 11 gezeigt, die kollisionsresistente Hashfunktion h gleichzeitig eine Einwegfunktion ist.

Die Verwendung der Hashfunktion h macht die existentielle Fälschung aus Abschn. 12.5.4 unmöglich. Angenommen, der Fälscher wählt eine Signatur s. Dann muss er auch eine Nachricht m mit $h(m) = s^e \bmod n$ finden, um ein gültiges Nachricht-Signatur-Paar (m, s) zu erzeugen. Das kann er aber nicht, weil h kolliosionsresistent und damit, wie in Abschn. 11.1 gezeigt wurde, eine Einwegfunktion ist.

Auch der Chosen-Plaintext-Angriff aus Abschn. 12.5.4, der auf der Multiplikativität des RSA-Verfahrens beruht, wird bei Verwendung einer kollisionsresistenten Hashfunktion unmöglich. Um die Signatur einer Nachricht m zu fälschen, wurden in Abschn. 12.5.4 nämlich Nachrichten m_1 und m_2 konstruiert, so dass die Signatur von m das Produkt der Signaturen von m_1 und m_2 modulo n ist. Es ist nicht klar, wie dieser Angriff auf die modifizierte Signiermethode, die eine Hashfunktion benutzt, übertragen werden kann. Soll

die Signatur von $m \in \mathbb{Z}_2^n$ gefälscht werden, könnte ein Angreifer zum Beispiel Hashwerte h_1 und h_2 konstruieren, so dass das Produkt der Signaturen von h_1 und h_2 modulo n die Signatur von $h(m)$ ist. Dann müsste er aber Nachrichten m_1 und m_2 finden mit $h(m_1) = h_1$ und $h(m_2) = h_2$ und nach den Signaturen von m_1 und m_2 fragen. Einem Chosen-Message-Angreifer ist es nämlich nur möglich, Signaturen für Nachrichten zu erhalten aber nicht für deren Hashwerte. Aber m_1 und m_2 kann der Angreifer nicht finden, weil h eine Einwegfunktion ist.

Die bisherige Argumentation in diesem Abschnitt hat nur die Einwegeigenschaft der Hashfunktion benutzt. Wir haben aber verlangt, dass die Hashfunktion kollisionsresistent ist. Warum? Weil andernfalls folgender Chosen-Message-Angriff möglich wird. Angenommen, ein Angreifer kann eine Kollision von h erzeugen, also ein Paar verschiedener Nachrichten $m, m' \in \mathbb{Z}_2^n$ mit $h(m) = h(m')$. Da es sich um einen Chosen-Message-Angriff handelt, kann sich der Angreifer die Nachricht m signieren lassen. Er erhält also $s = h(m)^d \bmod n$. Dann gilt aber auch $s = h(m') \bmod n$. Also ist s auch eine gültige Signatur von m'. Eine existentielle Fälschung ist gelungen.

12.5.7 Wahl von p und q

Wer den öffentlichen RSA-Modul faktorisieren kann, ist in der Lage, den geheimen Exponenten d zu bestimmen und damit RSA-Signaturen von Alice zu fälschen. Daher müssen p und q so gewählt werden, dass n nicht faktorisiert werden kann. Wie das gemacht wird, wurde in Abschn. 8.3.5 beschrieben.

12.5.8 Sichere Verwendung

Eine Variante des RSA-Signaturverfahrens, die unter geeigneten Voraussetzungen sicher gegen Chosen-Message-Angriffe ist, findet man in [7]. Es ist in PKCS #1 ab Version 2.1 standardisiert (siehe [56]). Dies wird in Abschn. 12.10.2 genauer diskutiert.

12.6 Signaturen aus Public-Key-Verfahren

Die Konstruktion des RSA-Signaturverfahrens lässt sich mit jedem deterministischen Public-Key-Verschlüsselungsverfahren nachahmen, sofern Ver- und Entschlüsselung vertauschbar sind. Diese Bedingung ist für das RSA-Verfahren erfüllt, weil

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

gilt. Bei der Konstruktion des Verfahrens muss man wie beim RSA-Verfahren eine Redundanz- oder eine Hashfunktion verwenden. Wie dies genau zu geschehen hat, ist z. B. in der Norm ISO/IEC 9796 [36] festgelegt.

Neben dem RSA-Verfahren erfüllt zum Beispiel das Rabin-Verfahren diese Bedingung. In Übung 12.3 wird ein Rabin-Signaturverfahren konstruieert.

12.7 ElGamal-Signatur

Wie das ElGamal-Verschlüsselungsverfahren (siehe Abschn. 8.7) bezieht auch das ElGamal-Signaturverfahren seine Sicherheit aus der Schwierigkeit, diskrete Logarithmen in $(\mathbb{Z}/p\mathbb{Z})^*$ zu berechnen, wobei p eine Primzahl ist. Da im ElGamal-Verschlüsselungsverfahren die Verschlüsselung und die Entschlüsselung nicht einfach vertauschbar sind, kann man daraus nicht direkt ein Signaturverfahren machen, jedenfalls nicht so, wie in Abschn. 12.6 beschrieben. Das ElGamal-Signaturverfahren wird daher anders konstruiert als das entsprechende Verschlüsselungsverfahren.

12.7.1 Schlüsselerzeugung

Die Schlüsselerzeugung funktioniert genauso wie im ElGamal-Verschlüsselungsverfahren (siehe Abschn. 8.7.1).

Alice wählt einen Sicherheitsparameter k, eine k-Bit Primzahl p und eine Primitivwurzel g mod 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.

12.7.2 Signatur

Alice signiert eine Nachricht $m \in \mathbb{Z}_2^*$. Sie benutzt eine öffentlich bekannte, kollisionsresistente Hashfunktion

$$h : \mathbb{Z}_2^* \to \{1, 2, \ldots, p-2\}.$$

Alice wählt eine Zufallszahl $k \in \{1, 2, \ldots, p-2\}$, die zu $p-1$ teilerfremd ist. Sie berechnet

$$r = g^k \bmod p, \quad s = k^{-1}(h(m) - ar) \bmod (p-1). \tag{12.7}$$

Hierin bedeutet k^{-1} das Inverse von k modulo $p-1$. Die Signatur ist das Paar (r, s). Weil eine Hashfunktion benutzt wurde, benötigt ein Verifizierer neben der Signatur auch die Nachricht m. Er kann m nicht aus s ermitteln.

12.7.3 Verifikation

Bob, der Verifizierer, kennt die Nachricht m und ihre Signatur (r, s). Er verschafft sich den öffentlichen Schlüssel (p, g, A) von Alice. Genauso wie beim RSA-Signaturverfahren muss er sich von der Authentizität des öffentlichen Schlüssels von Alice überzeugen. Er verifiziert, dass

$$1 \leq r \leq p-1$$

ist. Falls diese Bedingung nicht erfüllt ist, weist Bob die Signatur zurück. Andernfalls überprüft er, ob die Kongruenz

$$A^r r^s \equiv g^{h(m)} \bmod p \tag{12.8}$$

erfüllt ist. Wenn ja, akzeptiert Bob die Signatur, sonst nicht.

Wir zeigen, dass die Verifikation funktioniert. Falls s gemäß (12.7) konstruiert ist, folgt

$$A^r r^s \equiv g^{ar} g^{kk^{-1}(h(m)-ar)} \equiv g^{h(m)} \bmod p \tag{12.9}$$

wie gewünscht. Ist umgekehrt (12.8) für ein Paar (r, s) erfüllt, und ist k der diskrete Logarithmus von r zur Basis g, so gilt nach Korollar 2.1

$$g^{ar+ks} \equiv g^{h(m)} \bmod p.$$

Weil g eine Primitivwurzel mod p ist, folgt daraus

$$ar + ks \equiv h(m) \bmod p - 1.$$

Ist k zu $p-1$ teilerfremd, folgt daraus (12.7). Es gibt dann keine andere Art, die Signatur s zu konstruieren.

Beispiel 12.9 Wie in Beispiel 8.14 wählt Alice $p = 23$, $g = 7$, $a = 6$ und berechnet $A = g^a \bmod p = 4$. Ihr öffentlicher Schlüssel ist dann $(p = 23, g = 7, A = 4)$. Ihr geheimer Schlüssel ist $a = 6$.

Alice will eine Nachricht m mit Hashwert $h(m) = 7$ signieren. Sie wählt $k = 5$ und erhält $r = 17$. Das Inverse von k mod $(p - 1 = 22)$ ist $k^{-1} = 9$. Daher ist $s = k^{-1}(h(m) - ar) \bmod (p-1) = 9 * (7 - 6 * 17) \bmod 22 = 3$. Die Signatur ist $(17, 3)$.

Bob will diese Signatur verifizieren. Er berechnet $A^r r^s \bmod p = 4^{17} * 17^3 \bmod 23 = 5$. Er berechnet auch $g^{h(m)} \bmod p = 7^7 \bmod 23 = 5$. Damit ist die Signatur verifiziert.

12.7.4 Die Wahl von p

Wer diskrete Logarithmen mod p berechnen kann, ist in der Lage, den geheimen Schlüssel a von Alice zu ermitteln und damit Signaturen zu fälschen. Dies ist die einzige bekannte

allgemeine Methode, ElGamal-Signaturen zu erzeugen. Man muss also p so groß wählen, dass die Berechnung diskreter Logarithmen nicht möglich ist. Eine Diskussion der angemessenen Wahl von p findet sich in Abschn. 8.6.4.

Es gibt dabei eine spezielle Situation, die vermieden werden muss. Wenn $p \equiv 3 \bmod 4$, die Primitivwurzel g ein Teiler von $p-1$ ist und wenn die Berechnung diskreter Logarithmen in der Untergruppe von $(\mathbb{Z}/p\mathbb{Z})^*$ der Ordnung g möglich ist, dann kann das ElGamal-Signaturverfahren gebrochen werden (siehe Übung 12.5). Diskrete Logarithmen in dieser Untergruppe kann man jedenfalls dann berechnen (mit dem Algorithmus von Pohlig-Hellman-Shanks-Pollard), wenn g nicht zu groß ist (siehe Abschn. 10.5). Da die Primzahl p häufig so gewählt wird, dass $p-1 = 2q$ für eine ungerade Primzahl q gilt, ist die Bedingung $p \equiv 3 \bmod 4$ oft erfüllt. Um obige Attacke zu vermeiden, wählt man also eine Primitivwurzel g, die $p-1$ nicht teilt.

12.7.5 Die Wahl von k

Wir zeigen, dass aus Sicherheitsgründen für jede neue Signatur ein neuer Exponent k gewählt werden muss. Dies ist ja bei zufälliger Wahl von k garantiert.

Angenommen, die Signaturen s_1 und s_2 der Nachrichten m_1 und m_2 werden mit demselben Exponenten k erzeugt. Dann ist der Wert $r = g^k \bmod p$ für beide Signaturen gleich. Also gilt

$$s_1 - s_2 \equiv k^{-1}(h(m_1) - h(m_2)) \bmod (p-1).$$

Hieraus können wir die Zahl k bestimmen, wenn $h(m_1)-h(m_2)$ invertierbar modulo $p-1$ ist. Aus $k, s_1, r, h(m_1)$ lässt sich der geheime Schlüssel a berechnen. Es ist nämlich

$$s_1 = k^{-1}(h(m_1) - ar) \bmod (p-1)$$

und daher

$$a \equiv r^{-1}(h(m_1) - ks_1) \bmod (p-1).$$

12.7.6 Existentielle Fälschungen

Für die Sicherheit des Verfahrens ist es erforderlich, dass tatsächlich eine Hashfunktion verwendet wird und nicht die Nachricht m direkt signiert wird. Sonst ist eine existentielle Fälschung möglich, die im Folgenden beschrieben wird.

Wenn die Nachricht m ohne Hashfunktion signiert wird, lautet die Verifikationskongruenz

$$A^r r^s \equiv g^m \bmod p.$$

Wir zeigen, wie wir r, s, m wählen können, damit diese Kongruenz erfüllt ist. Man wählt zwei ganze Zahlen u, v mit $\gcd(v, p-1) = 1$. Dann setzt man

$$r = g^u A^v \bmod p, \quad s = -rv^{-1} \bmod (p-1), \quad m = su \bmod (p-1).$$

Mit diesen Werten gilt die Kongruenz

$$A^r r^s \equiv A^r g^{su} A^{sv} \equiv A^r g^{su} A^{-r} \equiv g^m \bmod p.$$

Das ist die Verifikationskongruenz.

Bei Verwendung einer kryptographischen Hashfunktion können wir auf die beschriebene Weise nur Signaturen von Hashwerten erzeugen. Wir können aber dazu nicht den passenden Klartext zurückgewinnen, weil die Hashfunktion eine Einwegfunktion ist.

Wie beim RSA-Verfahren kann das beschriebene Problem auch durch Einführung von Redundanz gelöst werden.

Auch die Bedingung $1 \leq r \leq p-1$ ist wichtig, um existentielle Fälschungen zu vermeiden. Wird diese Größenbeschränkung nicht gefordert, so können wir aus bekannten Signaturen neue Signaturen konstruieren. Sei (r, s) die Signatur einer Nachricht m. Sei m' eine weitere Nachricht, die signiert werden soll. Wir berechnen

$$u = h(m')h(m)^{-1} \bmod (p-1).$$

Hierbei wird vorausgesetzt, dass $h(m)$ invertierbar ist mod $p-1$. Wir berechnen weiter

$$s' = su \bmod (p-1)$$

und mit Hilfe des chinesischen Restsatzes ein r' mit

$$r' \equiv ru \bmod (p-1), \quad r' \equiv r \bmod p. \tag{12.10}$$

Die Signatur von m' ist (r', s'). Wir zeigen, dass die Verifikation mit dieser Signatur funktioniert. Tatsächlich gilt

$$A^{r'}(r')^{s'} \equiv A^{ru} r^{su} \equiv g^{u(ar+ks)} \equiv g^{h(m')} \bmod p.$$

Wir zeigen auch, dass $r' \geq p$ gilt, also den Test $1 \leq r' \leq p-1$ verletzt. Einerseits gilt

$$1 \leq r \leq p-1, \quad r \equiv r' \bmod p. \tag{12.11}$$

Andererseits ist

$$r' \equiv ru \not\equiv r \bmod p-1. \tag{12.12}$$

Das liegt daran, dass $u \equiv h(m')h(m)^{-1} \not\equiv 1 \bmod p-1$ gilt, weil h kollisionsresistent ist. Aus (12.12) folgt $r \neq r'$ und aus (12.11) folgt also $r' \geq p$.

12.7.7 Performanz

Die Erzeugung einer ElGamal-Signatur erfordert eine Anwendung des erweiterten euklidischen Algorithmus zur Berechnung von $k^{-1} \bmod p-1$ und eine modulare Exponentiation mod p zur Berechnung von $r = g^k \bmod p$. Beide Berechnungen können sogar als

Vorberechnungen durchgeführt werden. Sie hängen nicht von der zu signierenden Nachricht ab. Bei einer solche Vorberechnung müssen die Resultate aber sicher gespeichert werden. Die aktuelle Signatur erfordert dann nur noch zwei modulare Multiplikationen und ist damit extrem schnell.

Die Verifikation einer Signatur erfordert drei modulare Exponentiationen. Das ist deutlich teurer als die Verifikation einer RSA-Signatur. Die Verifikation kann beschleunigt werden, wenn man die Verifikationskongruenz (12.8) durch die neue Verifikationskongruenz

$$g^{-h(m)} A^r r^s \equiv 1 \bmod p$$

ersetzt und die modularen Exponentiationen auf der linken Seite simultan durchführt. Dies wird in Abschn. 2.13 erläutert. Aus Satz 2.16 folgt, dass die Verifikation höchstens $5 + t$ Multiplikationen und $t - 1$ Quadrierungen mod p erfordert, wobei t die binäre Länge von p ist. Das ist nur unwesentlich mehr als eine einzige Potenzierung.

12.7.8 Sichere Verwendung

Eine Variante des ElGamal-Signaturverfahrens, die unter geeigneten Voraussetzungen sicher gegen Chosen-Message-Angriffe ist, findet man in [57]. Dies wird in Abschnit 12.10.3 genauer diskutiert.

12.7.9 Verallgemeinerung

Wie das ElGamal-Verschlüsselungsverfahren lässt sich auch das ElGamal-Signaturverfahren in beliebigen zyklischen Gruppen implementieren, deren Ordnung bekannt ist aber in denen es schwer ist, diskrete Logarithmen zu berechnen. Solche Gruppen werden in Kap. 13 beschrieben. Das ist ein großer Vorteil des Verfahrens. Die Implementierung des Verfahrens kann aus obiger Beschreibung leicht abgeleitet werden. Natürlich müssen dieselben Sicherheitsvorkehrungen getroffen werden wie beim ElGamal-Signaturverfahren in $(\mathbb{Z}/p\mathbb{Z})^*$.

12.8 Der Digital Signature Algorithm (DSA)

Der Digital Signature Algorithm (DSA) wurde 1991 vom US-amerikanischen National Institute of Standards and Technology (NIST) vorgeschlagen und später von NIST zum Standard erklärt. Es beruht auf der ElGamal-Signaturvariante von Claus Schnorr [66]. Der DSA ist eine effizientere Variante des ElGamal-Verfahrens. Im DSA-Verfahren wird die Anzahl der modularen Exponentiationen bei der Verifikation von drei auf zwei reduziert und die Länge der Exponenten deutlich verkleinert. Außerdem ist die Parameterwahl viel

genauer vorgeschrieben als beim ElGamal-Verfahren. Inzwischen hat der Standard mehrere Revisionen durchlaufen. Wir verwenden hier die Revision von 2013 [30]

12.8.1 Schlüsselerzeugung

Der DSA-Schlüsselerzeugungsalgorithmus verwendet Parameterpaare $(k, l) \in \{(1024, 160), (2048, 224), (2048, 256), (3072, 256)\}$. Er erzeugt eine l-Bit Primzahl q und eine k-Bit Primzahl p, für die $q \mid p - 1$ gilt. Die Bedingung $q \mid (p - 1)$ impliziert, dass die Gruppe $(\mathbb{Z}/p\mathbb{Z})^*$ Elemente der Ordnung q besitzt (siehe Theorem 2.25). Ein solches Element wird nun konstruiert. Dazu bestimmt der Algorithmus $x \in \{2, \ldots p - 1\}$ mit der Eigenschaft

$$g = x^{(p-1)/q} \bmod p \neq 1. \tag{12.13}$$

Die Ordnung der Restklasse $g + p\mathbb{Z}$ ist dann q. Die Zahl x kann solange zufällig gewählt werden, bis (12.13) zutrifft. Zuletzt wählt der Algorithmus eine Zahl a zufällig in der Menge $\{1, 2, \ldots, q - 1\}$ und berechnet

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

Der öffentliche DSA-Schlüssel ist (p, q, g, A). Der private Signierschlüssel ist a. Man beachte, dass $A + p\mathbb{Z}$ zu der von $g + p\mathbb{Z}$ erzeugten Untergruppe (der Ordnung q) gehört. Die Untergruppe hat ungefähr 2^k Elemente. Die Ermittlung des geheimen Schlüssels erfordert die Berechnung diskreter Logarithmen in dieser Untergruppe. Wir werden im Abschnitt über die Sicherheit des Verfahrens darauf eingehen, wie schwierig die Berechnung diskreter Logarithmen in dieser Untergruppe ist.

12.8.2 Signatur

Signiert wird die Nachricht m. Dazu verwendet der Signieralgorithmus eine öffentlich bekannte kollisionsresistente Hashfunktion

$$h : \mathbb{Z}_2^* \to \{1, 2, \ldots, q - 1\}.$$

Danach wählt der Algorithmus eine Zufallszahl $k \in \{1, 2, \ldots, q - 1\}$, berechnet

$$r = (g^k \bmod p) \bmod q \tag{12.14}$$

und setzt

$$s = k^{-1}(h(m) + ar) \bmod q. \tag{12.15}$$

Hierbei ist k^{-1} das Inverse von k modulo q. Die Signatur ist (r, s).

12.8.3 Verifikation

Der Verifikationsalgorithmus soll die Signatur (r, s) der Nachricht m verifizieren. Er erhält m, (r, s), den Verifikationsschlüssel (p, q, g, A) und kennt die Hashfunktion h. Er verifiziert, dass

$$1 \leq r \leq q-1 \text{ und } 1 \leq s \leq q-1 \tag{12.16}$$

gilt. Ist eine dieser Bedingungen verletzt, ist die Signatur ungültig und wird zurückgewiesen. Andernfalls verifiziert der Algorithmus, dass

$$r = \left(\left(g^{(s^{-1}h(m)) \bmod q} A^{(rs^{-1}) \bmod q}\right) \bmod p\right) \bmod q \tag{12.17}$$

gilt. Ist die Signatur gemäß (12.14) und (12.15) korrekt konstruiert, so ist (12.17) tatsächlich erfüllt. Dann gilt nämlich

$$g^{(s^{-1}h(m)) \bmod q} A^{(rs^{-1}) \bmod q} \equiv g^{s^{-1}(h(m)+ra)} \equiv g^k \bmod p,$$

woraus (12.17) unmittelbar folgt.

12.8.4 Performanz

Das DSA-Verfahren ist sehr eng mit dem ElGamal-Signaturverfahren verwandt. Wie im ElGamal-Signaturverfahren kann die Erzeugung von Signaturen durch Vorberechnungen wesentlich beschleunigt werden.

Die Verifikation profitiert von zwei Effizienzsteigerungen. Sie benötigt nur noch zwei modulare Exponentiationen mod p, während im ElGamal-Verfahren drei Exponentiationen nötig sind. Dies ist aber nicht so bedeutend angesichts der Möglichkeit, die Verifikation durch simultane Exponentiation auszuführen (siehe die Abschn. 12.7.7 und 2.13). Bedeutender ist, dass die Exponenten in allen modularen Exponentiationen nur l Bit lang sind, $l \in \{160, 224, 256\}$, während im ElGamal-Verfahren die Exponenten genauso lang sind wie der Modul p. Diese Verkürzung beschleunigt die Berechnungen erheblich.

12.8.5 Sicherheit

Wie im ElGamal-Verfahren muss für jede neue Signatur ein neues k gewählt werden (siehe Abschn. 12.7.5). Außerdem ist die Verwendung einer Hashfunktion und die Überprüfung der Bedingungen in (12.16) unbedingt nötig. Wird keine Hashfunktion verwendet oder die erste Bedingung nicht überprüft, ergibt sich die Möglichkeit von existentiellen Fälschungen wie in Abschn. 12.7.6 beschrieben.

Wenn ein Angreifer diskrete Logarithmen in der von $g + p\mathbb{Z}$ erzeugten Untergruppe H von $(\mathbb{Z}/p\mathbb{Z})^*$ berechnen kann, so ist er in der Lage, den privaten Schlüssel a zu be-

stimmen und damit Signaturen seiner Wahl zu fälschen. Das ist bis heute auch die einzige bekannte Möglichkeit, DSA-Signaturen zu fälschen. Einen wesentlichen Effizienzvorteil bezieht das DSA-Verfahren daraus, dass die Berechnungen nicht mehr in der gesamten Gruppe $(\mathbb{Z}/p\mathbb{Z})^*$ ausgeführt werden, sondern in einer wesentlich kleineren Untergruppe. Es stellt sich also die Frage, ob dieses DL-Problem einfacher ist als das allgemeine DL-Problem.

Grundsätzlich sind zwei Möglichkeiten bekannt, diskrete Logarithmen zu berechnen.

Wir können Index-Calculus-Verfahren in $\mathbb{Z}/p\mathbb{Z}$ anwenden (siehe Abschn. 10.6). Es ist aber nicht bekannt, wie solche Verfahren einen Vorteil daraus ziehen können, dass ein diskreter Logarithmus in einer Untergruppe zu berechnen ist. Tatsächlich ist der Berechnungsaufwand genauso groß, wie wenn wir einen diskreten Logarithmus berechnen würden, dessen Basis eine Primitivwurzel modulo p ist.

Wir können auch generische Verfahren verwenden, die in allen endlichen abelschen Gruppen funktionieren. Die besten bekannten Verfahren von Shanks (siehe Abschn. 10.3) oder Pollard (siehe Abschn. 10.4) benötigen mehr als $\sqrt{q}$ Operationen in $(\mathbb{Z}/p\mathbb{Z})^*$, um diskrete Logarithmen in der Untergruppe der Ordnung q zu berechnen. Die Größe von q ist so gewählt, dass das unmöglich ist.

12.9 Das Merkle-Signaturverfahren

Die Sicherheit des RSA-Signaturverfahrens beruht auf der Schwierigkeit des Faktorisierungsproblems. Dagegen erfordert die Sicherheit der ElGamal oder DSA-Signaturverfahren, dass das Diskreter-Logarithmus-Problem (DLP) in Einheitengruppen endlicher Körper und in der Punktgruppe von elliptischen Kurven über endlichen Körpern schwer ist. Es ist aber keinewegs sicher, dass diese Probleme schwierig bleiben. Es ist zum Beispiel bekannt, dass Quantencomputer beide Probleme in Polynomzeit lösen können (siehe [69]). Daher ist es nötig, praktikable Alternativen bereitzustellen.

Das Lamport-Diffie-Einmal-Signaturverfahren (LD-OTS), das in Abschn. 12.3 beschrieben wurde, ist eine mögliche Alternative. Es ist sehr flexibel, denn es benötigt für seine sichere Implementierung nur eine Einwegfunktion. Solche Einwegfunktionen können zum Beispiel mit symmetrischen Verschlüsselungsverfahren oder kryptographischen Hashfunktionen erzeugt werden. Es gibt viele symmetrische Chiffren und kryptographische Hashfunktionen und es werden immer wieder neue entwickelt. Entsprechend läßt LD-OTS viele Instantiierungen zu. Sollte eine solche Instantiierung unsicher werden, kann sie durch eine andere ersetzt werden.

Leider ist LD-OTS nicht besonders praktikabel. Jedes Schlüsselpaar kann nämlich nur einmal verwendet werden. Dieses Problem löst das Merkle-Signaturverfahren [50]. Es verwendet einen binären Hashbaum, um die Gültigkeit vieler Einmal-Verifikationsschlüssel auf die Gültigkeit eines einzigen öffentlichen Schlüssels, der Wurzel des Hashbaumes, zurückzuführen. Das Merkle Verfahren kann also viele Signaturen erzeugen, die mit einem einzigen öffentlichen Schlüssel verifizierbar sind.

12.9.1 Initialisierung

Bei der Initialisierung des Merkle-Verfahrens wird eine Hashfunktion

$$h : \mathbb{Z}_2^* \to \mathbb{Z}_2^n$$

festgelegt und ein Einmal-Signaturverfahren gewählt. Es kann sich dabei um das Lamport-Diffie-Verfahren aus Abschn. 12.3 oder irgendein anderes Einmal-Signaturverfahren handeln. Statt eines dedizierten Einmal-Signaturverfahrens kann auch jedes andere Signaturverfahren als Einmal-Signaturverfahren benutzt werden.

Dann wird festgelegt, wie viele Signaturen mit einem einzigen öffentlichen Schlüssel verifizierbar sein sollen. Dazu wird eine natürliche Zahl H gewählt. Die Anzahl der verifizierbaren Signaturen ist $N = 2^H$.

Beispiel 12.10 Wir verwenden die Hashfunktion h, die folgendermaßen arbeitet: Der Hashwert sind die drei letzten Bits der Binärdarstellung der Quersumme der Dezimalzahl, die durch den zu hashenden String dargestellt wird. Ist diese Binärdarstellung zu kurz, werden führende Nullen ergänzt. Sei $m = 11000000100001$. Die entsprechende Dezimalzahl ist 12321. Die Quersumme dieser Zahl ist 9. Die Binärentwicklung von 9 ist 1001. Der Hashwert ist also $h(m) = 001$. Als Einmal-Signaturverfahren wird LD-OTS gewählt. Außerdem legen wir fest, dass mit einem öffentlichen Schlüssel vier Signaturen verifizierbar sein sollen. Die Höhe des Hashbaumes ist also $H = 2$.

12.9.2 Schlüsselerzeugung

Der Signierer wählt N Schlüsselpaare (x_i, y_i), $0 \leq i < N$, des verwendeten Einmal-Signaturverfahrens. Dabei ist jeweils x_i der Signierschlüssel und y_i der zugehörige Verifikationsschlüssel, $0 \leq i < N$. Als nächstes konstruiert der Signierer den Merkle-Hashbaum. Es handelt sich um einen binären Baum. Die Blätter dieses Baums sind die Hashwerte $h(y_i)$, $0 \leq i < N$, der Verifikationsschlüssel. Jeder Knoten im Baum, der kein Blatt ist, ist der Hashwert $h(k_l \circ k_r)$ seiner beiden Kinder k_l und k_r. Dabei ist k_l das linke Kind und k_r das rechte Kind. Der Merkle-Hashbaum wird in Abb. 12.1 gezeigt.

Der private Merkle-Signierschlüssel ist die Folge $(x_0, \ldots, x_{N-1})$ der gewählten Einmal-Signierschlüssel. Der öffentliche Schlüssel ist die Wurzel R des Merkle-Hashbaumes.

Beispiel 12.11 Wir setzen Beispiel 12.10 fort. Wir wählen also vier Paare (x_i, y_i), $0 \leq i < 4$. Dabei ist x_i jeweils ein Lamport-Diffie-Signierschlüssel und y_i ist der zugehörige Verifikationsschlüssel. Jedes x_i und jedes y_i besteht also aus sechs Bitstrings der Länge 3. Sie werden hier nicht explizit angegeben. Als nächstes wird der Hashbaum berechnet.

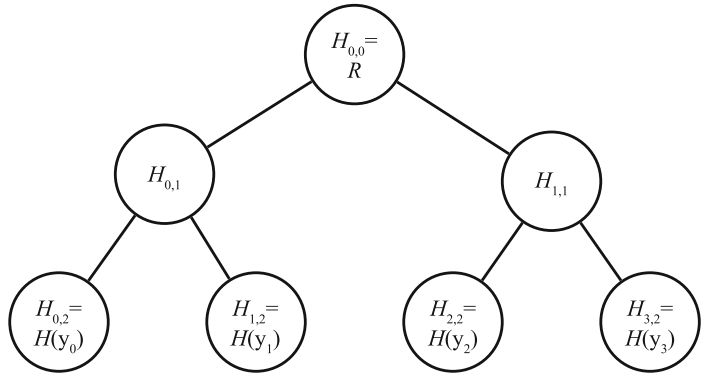

Abb. 12.1 Merkle-Hashbaum der Höhe $H = 2$

Die Knoten dieses Baums werden mit $h_{i,j}$ bezeichnet. Dabei ist i die Position (von links gezählt) und j die Tiefe des Knotens $h_{i,j}$ im Baum. Die Blätter des Baums seien

$$h_{0,2} = 110, h_{1,2} = 001, h_{2,2} = 010, h_{3,2} = 101.$$

Dann sind die Knoten auf Tiefe 1

$$h_{0,1} = h(h_{0,2} \circ h_{1,2}) = h(110001) = 101,$$
$$h_{1,1} = h(h_{2,2} \circ_{3,2}) = h(010101) = 011.$$

Die Wurzel ist

$$R = h_{0,0} = h(h_{0,1}||h_{1,1}) = h(101011) = 111.$$

Der geheime Schlüssel ist die Folge (x_0, x_1, x_2, x_3) der vier Lamport-Diffie-Signierschlüssel. Der öffentliche Schlüssel ist die Wurzel $R = 111$ des Hashbaumes.

12.9.3 Signatur

Eine Nachricht m soll signiert werden. Der Signierer verwendet einen Zähler i. Es handelt sich dabei um den Index des ersten noch nicht verwendeten Signierschlüssels. Der Wert von i ist der Zustand des Signiers. Das Merkle-Signaturverfahrens ist also im Gegensatz zu den bis jetzt besprochenen Signaturverfahren zustandsbehaftet.

Initial ist $i = 0$. Wenn $i > 2^H - 1$, kann keine Signatur mehr erzeugt werden. Der Signierer berechnet mit Einmal-Signierschlüssel x_i die Einmal-Signatur S der Nachricht m. Danach bestimmt er einen *Authentisierungspfad*. Er erlaubt es dem Verifizierer, die Gültigkeit des Verifikationsschlüssels y_i auf die Gültigkeit des öffentlichen Schlüssels R zurückzuführen. Der Authentisierungspfad für den Verifikationsschlüssel y_i ist eine Folge $(a_h, \ldots, a_1)$ von Knoten im Merkle-Hashbaum. Dieser Pfad ist durch folgende Eigenschaft charakterisiert. Bezeichne mit $(b_h, \ldots, b_1, b_0)$ den Pfad im Merkle-Hashbaum vom Blatt $b_h = h(y_i)$ zur Wurzel $b_0 = R$. Dann ist a_i, $h \geq i \geq 1$, der Knoten mit demselben

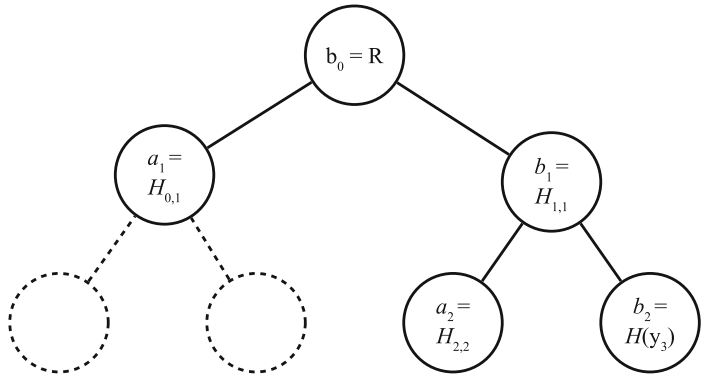

Abb. 12.2 Authentisierungspfad für das 4. Blatt.

Vater wie b_i. Dies wird in Abb. 12.2 illustriert. Die Signatur von m ist

$$s = (i, y_i, S, (a_h, \ldots, a_1)) .$$

Nachdem diese Signatur erstellt worden ist, wird der Zähler i um eins hochgesetzt.

Beispiel 12.12 Wir setzen das Beispiel 12.11 fort. Signiert werden soll die Nachricht m. Der Zähler i sei 3. Also wird der Signierschlüssel x_3 verwendet. Die Einmal-Signatur bezeichnen wir mit S. Wir bestimmen den Authentisierungspfad. Der Pfad vom Blatt $h(y_3) = h_{3,2}$ zur Wurzel R im Hashbaum ist $(b_2, b_1, b_0) = (h_{3,2}, h_{1,1}, h_{0,0})$. Der Authentisierungspfad ist $(a_2, a_1) = (h_{2,2}, h_{0,1})$. Die Signatur ist $(3, y_3, S, (a_2, a_1))$. Der Zähler wird auf 4 gesetzt.

12.9.4 Verifikation

Der Verifizierer erhält die Nachricht m und die Einmal-Signatur $(i, y_i, S, (a_h, \ldots, a_1))$. Zunächst verifiziert er die Signatur S unter Verwendung des Verifikationsschlüssels y_i. Wenn die Verifikation fehlschlägt, wird die Signatur zurückgewiesen. Ist die Verifikation erfolgreich, überprüft der Verifizierer die Gültigkeit des Verifikationsschlüssels y_i. Dazu verwendet er die Zahl i und den Authentisierungspfad $(a_h, \ldots, a_1)$. Er berechnet $b_h = h(y_i)$. Er weiß, dass die Knoten a_h und b_h im Merkle-Hashbaum denselben Vater haben. Er weiß nur noch nicht, ob a_h der linke oder der rechte Nachbar von b_h ist. Das liest er an i ab. Ist i ungerade, dann ist a_h der linke Nachbar von b_h. Andernfalls ist a_h der rechte Nachbar von b_h. Im ersten Fall bestimmt der Verifizierer den Knoten $b_{h-1} = h(a_h||b_h)$ im Pfad $(b_h, \ldots, b_1, b_0)$ vom Blatt $b_h = h(y_i)$ zur Wurzel $b_0 = R$ im Merkle-Hashbaum. Im zweiten Fall berechnet er $b_{h-1} = h(b_h \circ a_h)$. Die weiteren Blätter b_i, $h - 2 \geq i \geq 1$, werden analog berechnet. Dazu benötigt der Verifizierer die Binärentwicklung $i_0 \cdots i_h$ von i. Angenommen, der Verifizierer hat b_j berechnet, $h \geq j > 0$. Dann kann er b_{j-1} so

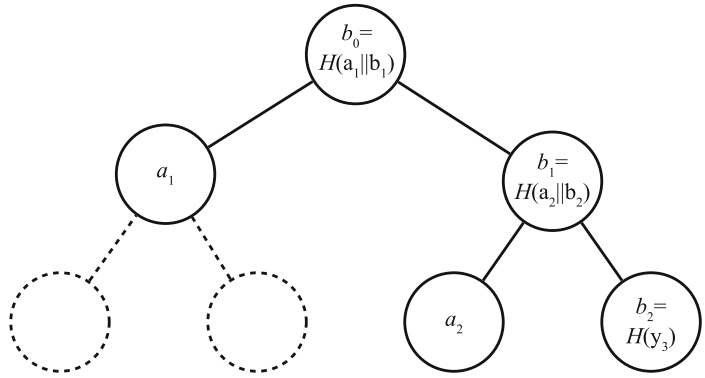

Abb. 12.3 Validierung eines Verifikationsschlüssels.

berechnen. Ist $i_j = 1$, so ist a_j der linke Nachbar von b_j, und es ist $b_{j-1} = h(a_j \circ b_j)$. Ist aber $i_j = 0$, dann ist a_j der rechte Nachbar von b_j und es gilt $b_{j-1} = h(b_j \circ a_j)$. Der Verifizierer akzeptiert die Signatur, wenn $b_0 = R$ ist und weist sie andernfalls zurück. Das Verifikationsverfahren ist in Abb. 12.3 illustriert.

Beispiel 12.13 Wir setzen das Beispiel 12.12 fort. Der Verifizierer kennt die Nachricht m und die Signatur $(3, y_3, S, (a_2 = 010, a_1 = 101))$. Er verifiziert die Signatur mit dem Verifikationsschlüssel y_3. Ist das erfolgreich, validiert er den Verifikationsschlüssel. Dazu bestimmt er die Binärentwicklung 11 von 3. Dann berechnet er den Knoten $b_2 = h(y_3) = 101$. Da das letzte Bit in der Binärentwicklung von $i = 3$ Eins ist, gilt $b_1 = h(a_2 \circ b_2) = h(010101) = 011$. Das erste Bit in der Binärdarstellung von $i = 3$ ist auch Eins. Darum ist $b_0 = h(a_1 \circ b_1) = h(101011) = 111$. Das ist tatsächlich der öffentliche Schlüssel. Also ist der öffentliche Schlüssel validiert und die Signatur wird akzeptiert.

12.9.5 Verbesserungen

Das ursprüngliche Merkle-Signaturverfahren in Kombination mit dem Lamport-Diffie-Einmal-Signaturverfahren hat eine Reihe von Nachteilen, die dazu geführt haben, dass das Verfahren nicht praktisch verwendet wurde. In den letzten Jahren sind aber viele Verbesserungen des Merkle-Verfahrens vorgeschlagen worden, die dieses Verfahren zu einem konkurrenzfähigen Signaturverfahren machen. Wir werden die Probleme und ihre Lösungen jetzt kurz beschreiben. Eine genauere Beschreibung mit entsprechenden Literaturangaben findet sich in [10].

Der Signierschlüssel im Merkle-Verfahren ist zu lang. Er besteht aus 2^H Einmal-Signierschlüsseln. Man kann ihn aber durch einen Zufallswert ersetzen, und daraus die Schlüssel mit einem Pseudozufallszahlengenerator erzeugen. Dies wurde von Coronado vorgeschlagen.

Im Lamport-Diffie-Einmal-Signaturverfahren wird jedes Bit einzeln signiert, und die Signatur jedes Bits ist ein Hashwert. Dadurch werden die Merkle-Signaturen sehr lang. Es sind aber in [10] verschiedene Vorschläge aufgeführt, Bits simultan zu signieren. Das macht die Signaturen deutlich kleiner. Aber je mehr Bits simultan signiert werden, desto mehr Hashwerte müssen bei der Signatur und der Verifikation berechnet werden. Die Anzahl der Hashwertberechnungen steigt dabei exponentiell mit der Anzahl der simultan signierten Bits. Daher ist die Möglichkeit, die Signatur auf diese Weise zu verkürzen ohne zu ineffizient zu werden, begrenzt.

Ein weiteres Effizienzproblem besteht darin, beim Signieren den Authentisierungspfad zu berechnen. Es ist natürlich möglich, dass der Signierer den ganzen Hashbaum speichert. Wenn die Höhe dieses Baums aber groß (≥ 10) wird, verbraucht der Hashbaum zu viel Platz. Dann können der Algorithmus von Szydlo oder seine Verbesserung von Dahmen verwendet werden. Dieser Algorithmus setzt die Kenntnis des Hashbaumes nicht voraus und benötigt nur Speicherplatz für $3H$ Knoten, wobei H die Höhe des Hashbaumes ist.

Auch wenn der Merkle-Baum bei Anwendung des Sydlo-Dahmen-Algorithmus nicht gespeichert werden muss, um Authentisierungspfade zu berechnen, muss er doch für die Erzeugung des öffentlichen Schlüssels vollständig berechnet werden. Ist die Höhe des Baums größer als 20, so wird das sehr langsam. Darum hat Coronado vorgeschlagen, statt eines Baums mehrere Bäume zu verwenden. Die können nach und nach berechnet werden. Diese Idee wurde von Dahmen und Vuillaume weiterentwickelt.

Wir haben schon erwähnt, dass das Merkle-Verfahren zustandsbehaftet ist, weil sich der Signierer den Index des ersten noch nicht verwendeten Signierschlüssels merken muss. Das hat den Nachteil, dass ein Signierer nicht einfach mehrere unabhängige Signiereinheiten verwenden kann. Diese müssten ihre Zustände synchronisieren. Eine zustandslose Variante findet sich in [11]. Der Vorteil des zustandsbehafteten Merkle-Verfahrens ist aber, dass alle schon verwendeten Signierschlüssel gelöscht werden können. Erhält ein Angreifer Zugang zu einer Signiereinheit und kann den geheimen Schlüssel auslesen, bekommt er keine Information über schon verwendete Signierschlüssel und kann deshalb alte Signaturen nicht nachträglich fälschen. Man nennt diese Eigenschaft *Vorwärtssicherheit*.

12.10 Sicherheitsmodelle

In diesem Abschnitt stellen wir formale Sicherheitsmodelle für digitale Signaturen vor. Sie haben Änlichkeit mit den Sicherheitsmodellen für Public-Key-Verschlüsselungsverfahren aus Abschn. 8.5. Wir diskutieren dann, inwieweit die in diesem Kapitel beschriebenen Signaturverfahren in den beschriebenen Modellen sicher sind.

12.10.1 Grundlagen

Sei $\mathbf{S} = (\mathbf{K}, \mathbf{M}, \mathbf{S}, \mathbf{KeyGen}, \mathbf{Sign}, \mathbf{Ver})$ ein digitales Signaturverfahren. Ein Angreifer A auf $\mathbf{S}$ ist ein probabilistischer Algorithmus A. Seine Eingabe ist ein Sicherheitsparame-

ter $k \in \mathbb{N}$ in unärer Darstellung 1^k und ein öffentlicher Schlüssel e aus der Menge $\mathbf{Pub}(k)$. Ziel des Angreifers ist zum Beispiel eine existentielle Fälschung. Der Angreifer versucht also, ein Paar (m, s) zu erzeugen, dessen erste Komponente eine Nachricht m und dessen zweite Komponente eine gültige Signatur $s \in \mathbf{S}(k)$ von m ist. Es muss also $1 \leftarrow \mathbf{Ver}(1^k, e, m, s)$ gelten. Es kann aber auch das Ziel des Angreifers sein, Signaturen von vorgegebenen Nachrichten zu fälschen. Dann erweitern wir seine Eingabe um die Nachricht, deren Signatur gefälscht werden soll. Die Laufzeit von Angreifern auf Signaturverfahren ist genauso definiert, wie die Laufzeit von Angreifern gegen die Sicherheit von Verschlüsselungsverfahren.

In Abschn. 12.4.2 wurde zwischen No-Message- und Chosen-Message-Angriffen unterschieden. Wenn der Angreifer einen No-Message-Angriff ausführt, muss er die existentielle Fälschung ohne weitere Hilfe durchführen. Bei einem Chosen-Message-Angriff hat der Angreifer Zugriff auf ein Orakel $\mathbf{Sign}_d$, das bei Eingabe einer Nachricht m' eine gültige Signatur für m' zurückgibt. Wir schreiben $A^{\mathbf{Sign}_d}$, um deutlich zu machen, dass A Zugriff auf das Orakel $\mathbf{Sign}_d$ hat.

Beispiel 12.14 Sei $k \in \mathbb{N}$ und $(n, e) \in \mathbf{K}(k)$, also ein öffentlicher RSA-Schlüssel zum Sicherheitsparameter k. Angreifer 12.1 führt den Angriff aus Abschn. 12.5.4 auf das RSA-Verfahren aus. Angreifer 12.2 führt den Chosen-Message-Angriff aus Abschn. 12.5.4 aus. Beide erzeugen eine existentielle Fälschung.

Angreifer 12.1 ($A(1^k, (n, e))$)

(No-Message Angreifer auf einfache RSA-Signaturen)

Wähle $s \in \mathbb{Z}_n$
$m \leftarrow s^e \bmod n$
return (m, s)

Angreifer 12.2 ($A^{\mathbf{Sign}_d}(1^k, (n, e), m)$)

(Chosen-Message-Angreifer auf einfache RSA-Signaturen)

Wähle $m_1 \in \mathbb{Z}_n$ mit $m_1 \neq m$ und $\gcd(m_1, n) = 1$
$m_2 \leftarrow m m_1^{-1} \bmod n$
$s_1 \leftarrow \mathbf{Sign}_d(m_1)$
$s_2 \leftarrow \mathbf{Sign}_d(m_2)$
$s \leftarrow s_1 s_2 \bmod n$
return s

Der Erfolg von No-Message- und Chosen-Message-Angreifern, deren Ziel eine existentielle Fälschung ist, wird mit Hilfe der Experimente 12.1 und 12.2 definiert. Entsprechende Experimente können für Angreifer definiert werden, deren Ziel die Fälschung von gegebenen Nachrichten ist.

Experiment 12.1 ($\mathbf{Exp}_{\mathbf{S}}^{\mathbf{nma}}(A, k)$)

(Experiment, das über den Erfolg eines No-Message-Angreifers auf das Signaturverfahren **S** entscheidet)

$(e, d) \leftarrow \mathbf{KeyGen}(1^k)$

$(m, s) \leftarrow A(1^k, e)$

$b \leftarrow \mathbf{Ver}(1^k, e, m, s)$

return b

Experiment 12.2 ($\mathbf{Exp}_{\mathbf{S}}^{\mathbf{cma}}(A, k)$)

(Experiment, das über den Erfolg eines Chosen-Message-Angreifers auf das Signaturverfahren **S** entscheidet)

$(e, d) \leftarrow \mathbf{KeyGen}(1^k)$

$(m, s) \leftarrow A^{\mathbf{Sign}_d}(1^k, e)$

$b \leftarrow \mathbf{Ver}(1^k, e, m, s)$

return b

Für $k \in \mathbb{N}$ ist der Vorteil eines No-Message-Angreifers A auf das Signaturverfahren **S**

$$\mathbf{Adv}_{\mathbf{S}}^{\mathrm{nma}}(A, k) = \Pr[\mathrm{Exp}_{\mathbf{Sign}}^{\mathrm{nma}}(A, k) = 1]. \tag{12.18}$$

Entsprechend ist der Vorteil eines Chosen-Message-Angreifers

$$\mathbf{Adv}_{\mathbf{S}}^{\mathrm{cma}}(A, k) = \Pr[\mathrm{Exp}_{\mathbf{Sign}}^{\mathrm{cma}}(A, k) = 1]. \tag{12.19}$$

Beispiel 12.15 Die Angreifer 12.1 und 12.2 haben den Vorteil 1, weil sie immer eine korrekte existentielle Fälschung ausgeben.

Jetzt definieren wir die Sicherheit von Signaturverfahren.

Definition 12.2 Seien $T : \mathbb{N} \to \mathbb{N}$ und $\varepsilon : \mathbb{N} \to [0, 1]$ Funktionen. Ein Signaturverfahren **S** heißt (T, ε)-sicher gegen No-Message/Chosen-Message-Angriffe, wenn für alle $k \in \mathbb{N}$ und alle No-Message/Chosen-Message-Angreifer A, die höchstens Laufzeit $T(k)$ haben, gilt: $Adv_{\mathbf{S}}^{\mathrm{nma/cma}}(A, k) < \varepsilon(k)$.

Wir definieren auch asymptotische Sicherheit von Signaturverfahren.

Definition 12.3 Ein Signaturverfahren **S** heißt asymptotisch sicher gegen No-Message/Chosen-Message-Angriffe, wenn für alle $c > 0$ und alle polynomiell beschränkten No-Message/Chosen-Message-Angreifer A gilt: $Adv_{\mathbf{S}}^{\text{nma/cma}}(A, k) = \mathrm{O}(1/k^c)$.

Beispiel 12.16 Das einfache RSA-Signaturverfahren ist nicht gegen No-Message-Angriffe sicher, weil es den Angreifer 12.1 gibt. Daher ist das einfache RSA-Signaturverfahren auch nicht gegen Chosen-Message-Angriffe sicher.

Es gibt bis jetzt kein Signaturverfahren, dessen Sicherheit bewiesen werden konnte. Es sind aber *Sicherheitsreduktionen* möglich. Dabei wird bewiesen, dass ein Signaturverfahren asymptotisch sicher ist, solange ein Berechnungsproblem nicht in Polynomzeit lösbar ist. Eine ausführlichere Diskussion von Sicherheitsreduktionen findet sich in Abschn. 8.5.3.

12.10.2 RSA

Wie verhält es sich mit der Sicherheit des RSA-Signaturverfahrens in den Modellen, die in Abschn. 12.10.1 entwickelt wurden? Wie in Beispiel 12.16 gezeigt, ist das einfache RSA-Signaturverfahren im No-Message- und Chosen-Message-Modell unsicher.

Die in Abschn. 12.5.8 erwähnte Variante *RSA-PSS* [7] des RSA-Signaturverfahrens, ist unter zwei Voraussetzungen sicher gegen Chosen-Message-Angriffe. Die erste Voraussetzung ist sie Gültigkeit der *RSA-Annahme*. Sie besagt, dass das *RSA-Problem* nicht in Polynomzeit zu lösen ist. Beim RSA-Problem ist ein öffentlicher RSA-Schlüssel (n, e) gegeben und $m \in \mathbb{Z}_n$. Gesucht ist $s \in \mathbb{Z}_n$ mit $m = s^e \bmod n$, also die e-te Wurzel von m modulo n. Die zweite Voraussetzung ist, dass die in RSA-PSS verwendete Hashfunktion ein *Zufallsorakel (random oracle)* ist. Es verhält sich so: Erhält es ein neues $m \in \mathbb{Z}_2^*$, so wählt es $h(m)$ zufällig und gleichverteilt. Wird aber $h(m)$ zum wiederholten Mal benötigt, so gibt das Zufallsorakel jedesmal denselben Wert zurück.

Kurz gesagt ist RSA-PSS also im Zufallsorakelmodell unter der RSA-Annahme sicher gegen Chosen-Message-Angriffe. Ist das ein überzeugendes Sicherheitsargument? Das wird aus verschiedenen Gründen kontrovers diskutiert. Reale Hashfunktionen sind nämlich keine Zufallsorakel. Außerdem ist die RSA-Annahme möglicherweise schwächer als die Annahme, dass Faktorisieren nicht in Polynomzeit möglich ist. Wünschenswert wäre ein Beweis, der die Sicherheit eines Signaturverfahrens auf die Schwierigkeit des Faktorisierungsproblems reduziert. Aber trotzdem ist die Sicherheitsreduktion für RSA-PSS ein wichtiges Sicherheitsindiz. Sie identifiziert zwei klar definierte Voraussetzungen dafür, dass RSA-PSS Chosen-Message-Angriffen widersteht. Beide Voraussetzungen sind in der Welt klassischer Computer bis jetzt unwidersprochen.

Tatsächlich haben Kryptographen nach der Veröffentlichung des Sicherheitsbeweises für RSA-PSS nach Signaturverfahren gesucht, die auf Zufallsorakel verzichten können. Ein solches Verfahren wurde 2000 von Ronald Cramer und Victor Shoup vorgestellt (siehe [23]). Die Sicherheit des Verfahrens setzt voraus, dass die *starke RSA-Annahme (strong*

RSA assumption) gilt. Sie besagt, dass das RSA-Problem auch dann noch schwer zu lösen ist, wenn der Angreifer nur den RSA-Modul bekommt aber den öffentlichen RSA-Exponenten $e \geq 3$ selbst wählen kann. Diese Annahme ist noch stärker als die RSA-Annahme. Außerdem ist das Cramer-Shoup-Verfahren deutlich ineffizienter als RSA-PSS. Darum wird dieses Verfahren in der Praxis nicht benutzt.

12.10.3 ElGamal

Für das ElGamal-Signaturverfahren sind keine Sicherheitsreduktionen bekannt. Für die ElGamal-Variante [57] von David Pointcheval und Jacques Stern aus dem Jahr 2000 ist das anders. Pointcheval und Stern beweisen, dass ihre Variante im Zufallsorakelmodell sicher ist, solange es nicht möglich ist, in Polynomzeit diskrete Logarithmen modulo α-schwerer Primzahlen zu berechnen, $0 < \alpha < 1$. Für solche Primzahlen p gilt $p - 1 = qr$ mit einer Primzahl q und $r \in \mathbb{N}$, wobei $r \leq p^{\alpha}$. Wird α klein genug gewählt, hat $p-1$ einen großen Primfaktor. Dies ist nötig, damit der Pohlig-Hellman-Algorithmus aus Abschn. 10.5 nicht anwendbar ist. Weil auch diese Sicherheitsreduktion das Zufallsorakelmodell verwendet, ist ihre Aussagekraft eingeschränkt wie in Abschn. 12.10.2 dargestellt.

Auch die Sicherheit der ElGamal-Variante von Claus Schnorr [66] kann im Zufallsorakelmodell auf die Schwierigkeit eines Diskreter-Logarithmus-Problems reduziert werden, nämlich in einer Untergruppe von Primzahlordnung q in der multiplikativen Gruppe eines Primkörpers $\mathbb{Z}/p\mathbb{Z}$.

12.10.4 Lamport-Diffie-Einmal-Signatur

Wir zeigen in diesem Abschnitt, dass das Lamport-Diffie-Einmalsignaturverfahren (LD-OTS), das in Abschn. 12.3 vorgestellt wurde, sicher gegen Chosen-Message-Angriffe ist, solange die Funktion h eine Einwegfunktion ist. Wir müssen diese Aussage präzisieren. Sicherheit von Signaturverfahren ist asymptotisch definiert. Daher verwenden wir statt dessen eine Familie $\mathbf{H} = \{h_n : \mathbb{Z}_2^n \to \mathbb{Z}_2^n : n \in \mathbb{N}\}$ von Einwegfunktionen. Ein *Invertierungsalgorithmus* A für H ist ein probabilistischer Algorithmus, der bei Eingabe von $n \in \mathbb{N}$ und $v \in \mathbb{Z}_2^n$ ein Urbild u von v unter h_n zurückgibt. Wir definieren, was es heißt, dass H die Einwegeigenschaft hat.

Definition 12.4 Die Funktionenfamilie $\mathbf{H}$ hat die *Einwegeigenschaft*, wenn für alle $c > 0$ und alle Invertierungsalgorithmen A für H gilt: die Erfolgswahrscheinlichkeit $\Pr(A, n)$ von A bei Eingabe von n ist $\mathrm{O}(1/n^c)$.

Als nächstes beschreiben wir, wie Chosen-Message-Angreifer für Einmal-Signaturen arbeiten. Ein Fälscher kennt einen LD-OTS-Verifikationsschlüssel y. Er versucht, eine Nachricht m und eine Signatur s für m zu finden, die sich mit y verifizieren lässt. Hilfsweise darf der Fälscher ein Orakel nach der Signatur für eine einzige Nachricht m fragen.

Im Chosen-Message-Modell für Mehrfach-Signaturverfahren darf der Fälscher viele Signaturen erfragen. Hier darf aber nur eine Signatur erfragt werden, weil es sich um ein Einmal-Signaturverfahren handelt und das Orakel bei einer weiteren Anfrage die Antwort verweigern müsste. Ein erfolgreicher Fälscher muss dann eine Nachricht und eine Signatur für die Nachricht ausgeben. Diese Nachricht darf aber nicht mit der übereinstimmen, für die der Fälscher bereits eine Orakel-Signatur erhalten hat. Der Vorteil solcher Angreifer ist analog zum Vorteil von Chosen-Message-Angreifern auf Mehrfach-Signaturverfahren definiert.

Jetzt reduzieren wir die Sicherheit von LD-OTS auf die Einwegeigenschaft von **H**. Angenommen, es gibt einen polynomiellen LD-OTS-Chosen-Message-Fälscher A. Unter Verwendung dieses Fälschers konstruieren wir einen Algorithmus, der ein Urbild für die von LD-OTS verwendete Einwegfunktion $h : \mathbb{Z}_2^n \to \mathbb{Z}_2^n$ berechnen kann. Dieser Algorithmus bekommt als Eingabe das Bild $v = h(u)$ eines zufällig gewählten $u \in \mathbb{Z}_2^n$. Seine Aufgabe ist es, ein Urbild von v unter h zu finden. Wir vewenden dieselben Bezeichnungen wie in Abschn. 12.3, wo LD-OTS eingeführt wurde. Den Schlüsselerzeugungsalgorithmus bezeichnen wir mit **KeyGen**.

Die Idee für den Invertierungsalgorithmus ist folgende. Er erzeugt einen LD-OTS-Schlüssel (x, y). Er wählt $(a, b) \in \mathbb{Z}_n \times \mathbb{Z}_2$ zufällig und ersetzt $y(a, b)$ durch u. Dann lässt er den Chosen-Message-Fälscher A laufen und beantwortet dessen Orakelanfrage folgendermaßen. Wenn der Fälscher nach der Signatur für ein $m' = (m'_1, \ldots, m'_n)$ fragt und $m'_b \neq a$ ist, antwortet der Algorithmus mit der richtigen Signatur. Die kann er erzeugen, weil er den Signaturschlüssel kennt mit Ausnahme des Urbildes von $y(a, b) = u$. Das wird aber für die Signatur nicht benötigt. Im Fall $m'_b \neq a$ liefert der Fälscher, falls er erfolgreich ist, also eine Nachricht $m = (m_1, \ldots, m_n) \neq m'$ und eine gültige Signatur s für m. Ist $m_b = a$, so ist die Komponente $s(b)$ ein Urbild von $u = y(a, b)$. Damit ist also das gesuchte Urbild gefunden. Ist $m'_b = a$, so kann der Algorithmus die Anfrage des Fälschers nicht beantworten. Dann gibt der Fälscher auch keine Fälschung aus, sondern $\perp$ und der Algorithmus bleibt erfolglos. Gibt der Fälscher die Signatur einer Nachricht m zurück und ist $m_b \neq a$, so ist der Algorithmus ebenfalls erfolglos. Wir bezeichnen den so modifizierten Fälscher mit $A^{(a,b)}$.

Algorithmus 12.1 (Invert^A (n, u))

(Algorithmus, der einen Chosen-Message-Angreifer auf LD-OTS benutzt, um die Hashfunktion h zu invertieren.

$$
\begin{aligned}
&(x, y) \leftarrow \mathbf{KeyGen}(1^n)\\
&a \xleftarrow{\$} \mathbb{Z}_2\\
&b \xleftarrow{\$} \{0, \ldots, n-1\}\\
&y(a, b) \leftarrow u\\
&r \leftarrow A^{(a,b)}(n, y)
\end{aligned}
$$

if $r \neq \perp, r = (m, s)$ und $m_b = a$ **then**
  return s_b
else
  return $\perp$
end if

Theorem 12.1 ist die Grundlage für den Sicherheitsbeweis für LD-OTS.

Theorem 12.1 *Sei A ein Chosen-Message-Angreifer gegen LD-OTS und sei* $n \in \mathbb{N}$. *Dann ist die Erfolgswahrscheinlichkeit von* Invert^A *mindestens* $\mathbf{Adv}^{\text{cma}}_{\text{LD-OTS}}(A, n)/2n$.

Beweis Invert^A ist erfolgreich, wenn der Fälscher eine gefälschte Signatur einer Nachricht $m = (m_1, \ldots, m_n)$ liefert und $m_b = a$ ist. Um diese Nachricht und die Signatur zu finden, darf der Fälscher das Signaturorakel nach einer Signatur fragen, muss es aber nicht. Angenommen der Fälscher fragt das Orakel nicht nach einer Signatur und liefert eine Fälschung. Dann ist $m_b = a$ mit Wahrscheinlichkeit $1/2$, weil a gleichverteilt zufällig gewählt wurde. Wenn der Fälscher das Orkal nach einer Signatur für eine Nachricht $m' = (m'_1, \ldots, m'_n)$ fragt, muss $m'_b = 1 - a$ sein, damit der Algorithmus, der das Orakel simuliert, richtig antworten kann. Die Wahrscheinlichkeit dafür ist $1/2$. Die Nachricht m, für die der Fälscher eine Signatur ausgibt, muss an wenigstens einer Stelle von m' verschieden sein. Die Wahrscheinlichkeit dafür, dass b eine dieser Stellen ist, beträgt wenigstens $1/n$. Damit ist die Erfolgswahrscheinlichkeit des Algorithmus mindestens $\mathbf{Adv}_{LD-OTS}(A, n)/2n$. □

Korollar 12.1 *Das Lamport-Diffie-Einmal-Signaturverfahren ist asymptotisch sicher gegen Chosen-Message-Angriffe, solange die Funktionenfamilie* **H** *die Einwegeigenschaft hat.*

Beweis Angenommen, LD-OTS ist nicht asymptotisch sicher. Wir müssen zeigen, dass **H** nicht die Einwegeigenschaft hat.

Weil LD-OTS nicht asymptotisch sicher ist, gibt es ein $c > 0$ und einen Chosen-Message-Angreifer, dessen Vorteil nicht in $O(1/n^c)$ ist. Aus Theorem 12.1 folgt, dass die Erfolgswahrscheinlichkeit des Invertierungsalgorithmus Invert^A nicht in $O(1/n^{c+1})$ ist. Also hat **H** nicht die Einwegeigenschaft. □

12.10.5 Merkle-Verfahren

Wir skizzieren einen Beweis dafür, dass das Merkle-Verfahren für feste Baumhöhe H im No-Message-Modell sicher ist, solange das Einmal-Signaturverfahren im No-Message-

Modell sicher ist und die verwendete Hashfunktion kollisionsresistent ist. Ein entsprechender Beweis im Chosen-Message-Modell wird dem Leser als Übung überlassen.

Angenommen, es gibt einen Fälscher, der bei Eingabe eines öffentlichen Merkle-Schlüssels eine gültige Merkle-Signatur mit nicht vernachlässigbarer Wahrscheinlichkeit ε fälschen kann. Wir konstruieren einen Algorithmus, der diesen Fälscher verwendet, um entweder eine existentielle Fälschung des Einmal-Signaturverfahrens zu produzieren oder eine Kollision der verwendeten Hashfunktion zu finden.

Dieser Algorithmus bekommt als Eingabe einen Verifikationsschlüssel y für das Einmal-Signaturverfahren und die Hashfunktion h. Es ist seine Aufgabe, eine existentielle Fälschung für das Einmal-Signaturverfahren zu finden, die sich mit dem Verifikationsschlüssel y verifizieren lässt oder eine Kollision für h zu produzieren.

Der Algorithmus erzeugt zufällig $N = 2^H$ Einmal-Schlüsselpaare

$$((x_0, y_0) \ldots, (x_{N-1}, y_{N-1})).$$

Er wählt zufällig eine Position $c \in \{0, \ldots, N-1\}$ und ersetzt y_c durch den eingegebenen Einmal-Verifikationsschlüssel y. Dann konstruiert der Algorithmus den Merkle-Baum und bestimmt den öffentlichen Merkle-Schlüssel R. Als nächstes ruft der Algorithmus den Merkle-Fälscher mit dem öffentlichen Schlüssel R auf. Dieser antwortet (mit Wahrscheinlichkeit ε) mit einer Nachricht m und einer gültigen Signatur (i, y', s, A'). Hierbei bezeichnet y' den Einmal-Verifikationsschlüssel und A' den Authentisierungspfad. Man beachte, dass y' nicht unbedingt mit y_i übereinstimmen muss. Der Verifizierer kennt ja y_i nicht. Er muss sich bei der Verifikation nur davon überzeugen, dass die Einmal-Signatur mit dem angegebenen Verifikationsschlüssel y' funktioniert, und dass der Pfad von diesem Verifikationsschlüssel zum öffentlichen Schlüssel mit Hilfe des Authentierungspfads A' konstruierbar ist.

Im Folgenden schreiben wir $y = y_i$ und bezeichnen mit A den Authentisierungspfad für y im vom Algorithmus konstruierten Merkle Baum. Ist $(y, A) \neq (y', A')$, so kann der Algorithmus folgendermaßen eine Kollision der Hashfunktion finden. Konstruiert man aus (y, A) und (y', A') nach der Merkle-Konstruktion Pfade B und B' von den Blättern $h(y)$ und $h(y')$ zur Wurzel des Merkle-Baums, so führen beide Pfade zum öffentlichen Merkle-Schlüssel. Für B stimmt das, weil der Algorithmus den öffentlichen Merkle-Schlüssel entsprechend konstruiert hat. Für B' ist das richtig, weil der Fälscher eine gültige Signatur liefert, und daher die Verifikation der gefälschten Signatur erfolgreich ist. Da aber $(y, A) \neq (y', A')$ gilt, muss $(A, B) \neq (A', B')$ gelten. Also gibt es auf dem Weg von der Wurzel zu den Blättern einen ersten Knoten, der in beiden Pfaden gleich ist, der aber als Hashwert von zwei Knotenpaaren entsteht, die verschieden sind. Damit ist eine Kollision gefunden, die der Algorithmus zurückgibt. Im Fall $(y, A) \neq (y', A')$ ist der Algorithmus mit Wahrscheinlichkeit ε erfolgreich, wobei ε die Erfolgswahrscheinlichkeit des Fälschers ist.

Ist $(y, A) = (y', A')$ und ist $i = c$. Dann ist $y = y'$ der vorgegebene Einmal-Signaturschlüssel. Da s eine gültige Einmal-Signatur für das Dokument d ist, die sich

mit y verifizieren lässt, ist eine existentielle Fälschung des Einmal-Signaturverfahrens mit öffentlichem Schlüssel y gefunden. Im Fall $(y, A) = (y', A')$ ist die Erfolgswahrscheinlichkeit also ε/N.

Da einer der Fälle $(y, A) = (y', A')$ oder $(y, A) \neq (y', A')$ mit Wahrscheinlichkeit $\geq 1/2$ eintritt, ist die Erfolgswahrscheinlichkeit des Algorithmus insgesamt $\geq \varepsilon/2N = \varepsilon/2^{H+1}$. Diese Wahrscheinlichkeit ist nicht vernachlässigbar, weil H konstant ist und ε nicht vernachlässigbar ist.

12.11 Übungen

Übung 12.1 Berechnen Sie die RSA-Signatur (ohne Hashfunktion) von $m = 11111$ mit RSA-Modul $n = 28829$ und dem kleinstmöglichen öffentlichen Exponenten e.

Übung 12.2 Stellt die Low-Exponent-Attacke oder die Common-Modulus-Attacke ein Sicherheitsproblem für RSA-Signaturen dar?

Übung 12.3 Wie kann man aus dem Rabin-Verschlüsselungsverfahren ein Signaturverfahren machen? Beschreiben Sie die Funktionsweise eines Rabin-Signaturverfahrens und diskutieren Sie seine Sicherheit und Effizienz.

Übung 12.4 Berechnen Sie die Rabin-Signatur (ohne Hashfunktion) von $m = 11111$ mit dem Rabin-Modul $n = 28829$.

Übung 12.5 Im ElGamal-Signaturverfahren werde die Primzahl p, $p \equiv 1 \bmod 4$ und die Primitivwurzel $g \bmod p$ benutzt. Angenommen, $p - 1 = gq$ mit $q \in \mathbb{Z}$ und g hat nur kleine Primfaktoren. Sei A der öffentliche Schlüssel von Alice.

1. Zeigen Sie, dass sich eine Lösung z der Kongruenz $A^q = g^{qz} \bmod p$ effizient finden lässt.
2. Sei x ein Dokument und sei h der Hashwert von x. Zeigen Sie, dass $(q, (p-3)(h - qz)/2)$ eine gültige Signatur von x ist.

Übung 12.6 Sei $p = 130$. Berechnen Sie einen gültigen privaten Schlüssel a und den entsprechenden öffentlichen Schlüssel (p, g, A) für das ElGamal-Signaturverfahren.

Übung 12.7 Sei $p = 2237$ und $g = 2$. Der geheime Schlüssel von Alice ist $a = 1234$. Der Hashwert einer Nachricht m sei $h(m) = 111$. Berechnen Sie die ElGamal-Signatur mit $k = 2323$ und verifizieren Sie sie.

Übung 12.8 Angenommen, im ElGamal-Signaturverfahren ist die Bedingung $1 \leq r \leq p - 1$ nicht gefordert. Verwenden Sie die existentielle Fälschung aus Abschn. 12.7.6, um eine ElGamal-Signatur eines Dokumentes x' mit Hashwert $h(x') = 99$ aus der Signatur in Beispiel 12.7 zu konstruieren.

Übung 12.9 Es gelten dieselben Bezeichnungen wie in Übung 12.7. Alice verwendet das DSA-Verfahren, wobei q der größte Primteiler von $p-1$ ist. Sie verwendet aber $k=25$. Wie lautet die entsprechende DSA-Signatur? Verifizieren Sie die Signatur.

Übung 12.10 Erläutern Sie die existentielle Fälschung aus Abschn. 12.7.6 für das DSA-Verfahren.

Übung 12.11 Wie lautet die Verifikationskongruenz, wenn im ElGamal-Signaturverfahren s zu $s=(ar+kh(x)) \bmod p-1$ berechnet wird?

Übung 12.12 Modifizieren Sie die ElGamal-Signatur so, dass die Verifikation nur zwei Exponentiationen benötigt.

Übung 12.13 Beweisen Sie die Sicherheit des Merkle-Signaturverfahrens im Chosen-Message-Modell.