Hash-Funktionen

11

Eine Hash-Funktion ist ein wichtiges kryptografisches Primitiv, das in sehr vielen Protokollen eingesetzt wird. Hash-Funktionen berechnen aus einer gegebenen Nachricht eine Bitfolge mit fester Länge, die ein Repräsentant der Nachricht ist. Der so erzeugte Hash-Wert kann als ein Fingerabdruck der Nachricht betrachtet werden. Im Gegensatz zu allen anderen Kryptoalgorithmen, die bisher in diesem Buch eingeführt wurden, haben Hash-Funktionen keinen Schlüssel. Hash-Funktionen haben viele Anwendungen in der Kryptografie. Sie sind ein wichtiger Teil von Signaturverfahren und kryptografischen Prüfsummen (MAC), die in Kap. 12 besprochen werden. Hash-Funktionen haben darüber hinaus noch viele andere Einsatzmöglichkeiten, z. B. das Speichern von Passwörtern oder Schlüsselableitung.

In diesem Kapitel erlernen Sie

- warum Hash-Funktionen für digitale Signaturen benötigt werden,
- wichtige Eigenschaften von Hash-Funktionen,
- eine Diskussion über Sicherheit von Hash-Funktionen, wozu auch die Einführung des Geburtstagsparadoxons gehört,
- einen Überblick über Hash-Funktionen, die in der Praxis eingesetzt werden,
- wie die weit verbreitete Hash-Funktion SHA-1 funktioniert.

11.1 Motivation: Das Signieren langer Nachrichten

Hash-Funktionen, im Deutschen auch vereinzelt Streuwertfunktionen genannt, haben viele Anwendungen in der modernen Kryptografie. Am bekanntesten sind sie wahrscheinlich für ihre Rolle, die sie bei digitalen Signaturen spielen. In den vorherigen Kapiteln wurden digitale Signaturen vorgestellt, die auf RSA oder dem DLP basieren. Bei all diesen Verfahren ist der Klartext, der signiert wird, in der Größe begrenzt. Beispielsweise kann

C. Paar, J. Pelzl, *Kryptografie verständlich*, eXamen.press,
DOI 10.1007/978-3-662-49297-0_11

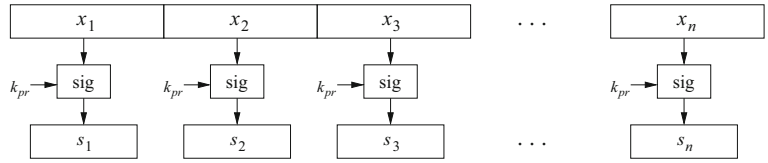

Abb. 11.1 Unsicherer Ansatz für das Signieren großer Nachrichten

für RSA die Nachricht nicht länger als der Modul sein. Dieser ist in der Praxis oft zwischen 1024 und 3072 Bit lang, d. h. zwischen 128 und 384 Byte – leider sind die meisten E-Mails schon länger. Allgemein lässt sich sagen, dass wir bisher die Tatsache ignoriert haben, dass die meisten Klartexte, die in der Praxis auftauchen, (viel) länger sind als die oben genannten Bitlängen. Daraus ergibt sich jetzt die Fragestellung, wie wir effizient Signaturen für lange Nachrichten berechnen können. Ein naheliegender Ansatz, der dem ECB-Modus für Blockchiffren ähnelt, ist wie folgt: Unterteile die Nachricht x in Blöcke x_i, die nicht länger sind als die Eingangslänge des Signaturverfahrens und signiere jeden Block einzeln. Dieser Ansatz ist in Abb. 11.1 dargestellt.

Bei diesem naiven Ansatz ergibt sich allerdings eine ganze Reihe Probleme:

Problem 1: Hoher Rechenaufwand Digitale Signaturen basieren auf asymmetrischen Algorithmen, die rechenintensiv sind, da sie z. B. modulare Exponentiation mit großen Zahlen erfordern. Auch wenn eine einzelne dieser Operationen schnell ist und wenig Energie erfordert (was bei mobilen Anwendungen wichtig ist), würde die Signatur von langen Nachrichten, z. B. Anhängen von E-Mails oder Videodateien, viel zu lange brauchen. Auch würde nicht nur der Sender eine lange Signatur berechnen müssen, auch der Empfänger müsste eine lange Signatur verifizieren.

Problem 2: Nachrichten-Overhead Der oben beschriebene naive Ansatz verdoppelt die Länge der zu sendenden Bits, denn nicht nur die Nachricht, sondern auch die Signatur, die die gleiche Länge hat, muss übermittelt werden. Beispielsweise hätte eine 1-MByte-Datei eine RSA-Signatur der Länge von einem MByte, sodass insgesamt zwei Megabyte übertragen werden müssen.

Problem 3: Sicherheitsprobleme Dies ist das schwerwiegendste Problem, das auftritt, wenn wir versuchen, eine lange Nachricht zu signieren, indem wir einzelne Nachrichtenblöcke individuell signieren. Aus dem Ansatz, der in Abb. 11.1 dargestellt ist, ergeben sich sofort neue Angriffsmöglichkeiten. Oskar könnte beispielsweise einzelne Nachrichten und die dazugehörigen Signaturen entfernen oder könnte Nachrichten und Signaturen umordnen. Er könnte auch aus alten Nachrichtenblöcken neue Nachrichten zusammenstellen. Obwohl Oskar keine Manipulationen innerhalb eines Blocks vornehmen kann, kann er die Nachricht im Ganzen manipulieren.

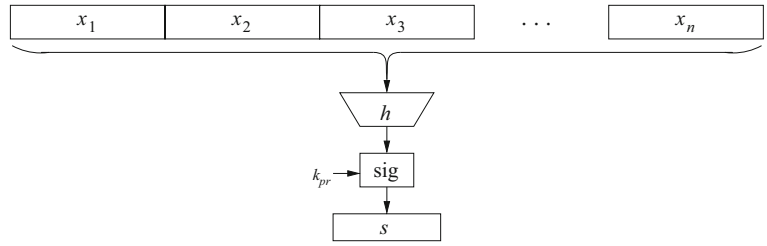

Abb. 11.2 Signieren einer langen Nachricht mithilfe einer Hash-Funktion

Aus dieser Diskussion folgt, dass wir eine *einzelne* kurze Signatur für eine Nachricht beliebiger Länge benötigen. Wir erreichen dieses Ziel mit einer Hash-Funktion, die einen Fingerabdruck einer Nachricht berechnet. Hiermit kann dann eine Signatur, wie in Abb. 11.2 dargestellt, errechnet werden.

Wenn eine solche Hash-Funktion existiert, kann man das folgende Basisprotokoll für digitale Signaturen ausführen. Hierbei möchte Bob eine digital signierte Nachricht an Alice senden.

Basisprotokoll für digitale Signaturen mit Hash-Funktionen

Alice		**Bob**
	$\xleftarrow{k_{\text{pub},B}}$	
		$z = h(x)$
		$s = \text{sig}_{k_{\text{pr},B}}(z)$
	$\xleftarrow{(x,s)}$	
$z' = h(x)$		
$\text{ver}_{k_{\text{pub},B}}(s, z') = \text{richtig/falsch}$		

Bob berechnet zunächst den Hash-Wert der Nachricht x und signiert den Hash-Wert z mit seinem privaten Schlüssel $k_{\text{pr},B}$. Auf der Empfängerseite berechnet Alice den Hash-Wert z' der empfangenen Nachricht x. Sie verifiziert die Signatur s mit Bobs öffentlichem Schlüssel $k_{\text{pub},B}$. Man beachte, dass sowohl die Signaturerzeugung als auch die Verifikation jeweils den Hash-Wert z als Eingang hat und nicht die eigentliche Nachricht. Daher kann gesagt werden, dass der Hash-Wert die Nachricht repräsentiert. Aus diesem Grund wird der Hash-Wert manchmal auch der Fingerabdruck der Nachricht genannt.

Bevor wir in den folgenden Abschnitten die Sicherheitseigenschaften von Hash-Funktionen diskutieren, überlegen wir uns, welches Eingang-Ausgang-Verhalten Hash-Funktionen aufweisen sollten. Sie sollten eine Nachricht x beliebiger Länge verarbeiten

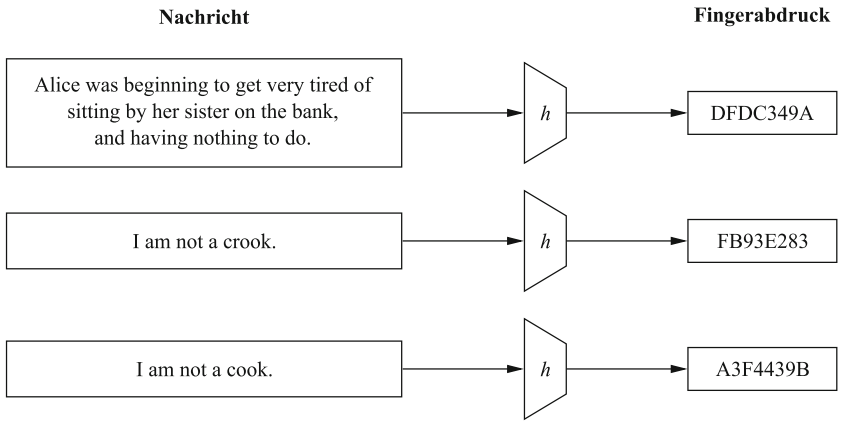

Abb. 11.3 Eingang- und Ausgangverhalten von Hash-Funktionen

können. Von daher sollte die Hash-Funktion sehr schnell zu berechnen sein. In der Praxis bedeutet dies, dass selbst lange Nachrichten mit einer Länge von mehreren Megabyte oder sogar Gigabyte schnell verarbeitet werden sollten. Eine weitere wünschenswerte Eigenschaft von Hash-Funktionen ist, dass der Ausgangswert eine feste Länge hat, der unabhängig von der Länge des Eingangs ist. In der Praxis verwendete Hash-Funktionen haben Ausgangslängen von 128–512 Bit. Weiterhin soll der gesamte berechnete Fingerabdruck von allen Eingangsbits abhängen. Hieraus folgt, dass selbst kleine Änderungen im Eingangswert zu einem vollständig unterschiedlichen Fingerabdruck führen sollen. Ein ähnliches Verhalten haben wir bei Blockchiffren beobachtet. Die Eigenschaften, die wir bisher diskutiert haben, sind in Abb. 11.3 symbolisch dargestellt.

11.2 Sicherheitseigenschaften von Hash-Funktionen

Wie eingangs schon erwähnt, haben Hash-Funktionen, im Gegensatz zu allen bisher vorgestellten Kryptoalgorithmen, keinen Schlüssel. Hieraus ergibt sich die Frage, ob es überhaupt Sicherheitseigenschaften gibt, die Hash-Funktionen erfüllen müssen. Da Hash-Funktionen Daten nicht ver- oder entschlüsseln, ist diese Frage durchaus berechtigt. Wie oft in der Kryptografie sind Fragestellungen zur Sicherheit sehr diffizil und es gibt eine Reihe von Angriffen, die Schwachstellen von Hash-Funktionen ausnutzen können. Es gibt drei zentrale Eigenschaften, die für die Sicherheit von Hash-Funktionen entscheidend sind:

1. Urbildresistenz (oder Einwegeigenschaft)
2. Schwache Kollisionsresistenz (oder zweite Urbildresistenz)
3. Starke Kollisionsresistenz (oder Kollisionsresistenz)

Abb. 11.4 veranschaulicht diese Eigenschaften, die wir in den folgenden Abschnitten diskutieren werden.