Tom St Denis / Simon Johnson

Kryptografie für Entwickler

Das erste umfassende Kryptografie-Handbuch für Software-Entwickler

- Verschlüsselungstechniken verstehen und anwenden
- Alles über die Anwendungsgebiete: Datenschutz und -sicherheit, Authentifizierung und Nachweisbarkeit
- Integercodierung, Public-Key-Algorithmen und AES in der Praxis

FRANZIS

Tom St Denis / Simon Johnson

Kryptografie für Entwickler

Tom St Denis / Simon Johnson

Kryptografie für Entwickler

Das erste umfassende Kryptografie-Handbuch für Software-Entwickler

- Verschlüsselungstechniken verstehen und anwenden
- Alles über die Anwendungsgebiete: Datenschutz und -sicherheit, Authentifizierung und Nachweisbarkeit
- Integercodierung, Public-Key-Algorithmen und AES in der Praxis

Bibliografische Information der Deutschen Bibliothek

Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie;
detaillierte Daten sind im Internet über http://dnb.ddb.de abrufbar.

This edition of **Cryptography for Developers** by **Tom St Denis** is published by arrangement with ELSEVIER Inc., a Delaware corporation having its principal place of business at 360 Park Avenue South, New York, NY 10010, USA

Programmleitung: Dr. Markus Stäuble
Satz und Übersetzung: G&U Language & Publishing Services GmbH
art & design: www.ideehoch2.de

ISBN 978-3-645-20543-6

Inhaltsverzeichnis

Vorwort

Da sind wir nun im Vorwort zu meinem zweiten Buch. Ich weiß gar nicht genau, was ich Ihnen hier mitteilen soll, außer dass dieses Buch viel dramatischer und fesselnder ist als das letzte. An dieser Stelle möchte ich noch nicht zu viel verraten, außer dass RSA hinter dem Rücken von MD5 eine Affäre mit SHA hat. Aber Spaß beiseite.

Diese Zeilen schreibe ich praktisch am Vorabend des Drucktermins. Ich kann es kaum erwarten, das fertige Produkt in den Händen zu halten, und hoffe sehr, mit diesem Buch meine selbst gesteckten Ziele erreicht zu haben. Es ist das Arbeitsergebnis fast eines ganzen Jahres, von Anfang 2006 bis beinahe in den November 2006 hinein. Ich habe viele Abende damit zugebracht, nach der Arbeit noch zu schreiben, und kann nur hoffen, dass dieser Text beim Zielpublikum gut ankommt. Die Arbeit hat Spaß gemacht, auch wenn sie zuweilen mühselig war, und hat sich wie schon bei meinem ersten Buch gelohnt.

Bevor ich Ihnen mehr über dieses Buch erzähle, sollte ich Ihnen erst einmal sagen, wer die Autoren sind. Der Text wurde größtenteils von mir, Tom St Denis, geschrieben, wobei ich Hilfe von meinem Co-Autor und Fachgutachter Simon Johnson bekommen habe. Ich bin ein Informatiker aus Ontario mit einer Leidenschaft für alles, was mit Kryptografie zusammenhängt. Insbesondere arbeite ich gern mit spezialisierter Hardware und Embedded Systems.

Meine Bekanntheit verdanke ich den LibTom-Projekten, über die Sie vielleicht auch auf dieses Buch aufmerksam geworden sind. Dabei handelt es sich um eine Reihe von kryptografischen und mathematischen Bibliotheken für verschiedene Probleme, mit denen Entwickler in der Praxis zu kämpfen haben. Sie wurden allerdings auch so geschrieben, dass man etwas daraus lernen kann. LibTomCrypt, mein erstes Projekt, ist das Produkt von nahezu fünf Jahren Arbeit. Diese Bibliothek unterstützt ziemlich viele nützliche kryptografische Primitiva und ist auch eine sehr gute Quelle für die Arbeit mit diesem Buch. Als nächstes dieser kryptografischen Projekte habe ich 2002 mit LibTomMath begonnen. Dabei handelt es sich um eine portierbare Mathematikbibliothek für den Umgang mit großen Integern. Sie hat als eine der mathematischen Standardbibliotheken für LibTomCrypt ein Zuhause gefunden und ist auch elementarer Bestandteil von Projekten wie Tcl und Dropbear. Um noch darüber hinaus zu gehen, habe ich TomsFastMath geschrieben, eine unglaublich schnelle und leicht zu portierende Mathe-Bibliothek für kryptografische Operationen.

All diese Projekte habe ich als »freie« Projekte geschrieben, was nicht nur bedeutet, dass sie kostenlos sind, sondern auch, dass es keine versteckten Bedingungen gibt. Sie sind ohne Wenn und Aber gemeinfrei. Für mich genügt es auch nicht, einfach nur den Code abzuliefern. Ich habe auch eine Dokumentation mitgeliefert, die erklärt, wie Sie die Projekte verwenden können. Doch damit nicht genug! Ich habe auch den Quellcode dokumentiert und bereinigt, sodass er auch zum Lernen geeignet ist. Mein erstes Projekt dieser Art war LibTomMath. 2003 habe ich das Buch *BigNum Math: Implementing Cryptographic Multiple Precision Arithmetic* (ISBN 1597491128) geschrieben, das 2006 bei Syngress erschienen ist. Dabei habe ich Code aus dem Projekt verwendet. Zusammen mit dem beigefügten Pseudocode können Sie daraus lernen, wie Sie ohne große Mühen mit großen Integern umgehen können.

Die LibTom-Projekte stehen unter einem einfachen Motto, das ich im Laufe der Jahre entwickelt habe:

Open Source. Open Academia. Open Mind.

Das bedeutet, dass wir durch die Bereitstellung von Quellcode mit einer nützlichen Dokumentation und Zusatzmaterial andere unterrichten und gegenüber neuen Ideen und Techniken aufgeschlossener machen können. Dieses Motto erweitert die Open-Source-Philosophie um eine didaktische Dimension. Beispielsweise ist es zwar schön und gut, dass GCC (GNU Compiler Collection) quelloffen ist, aber zum Lernen ist dieses Projekt trotzdem nicht geeignet. Aber genug davon; diese Gedankengänge gehören schon zu meinem nächsten Buch (das 2009 fällig sein dürfte).

Ich arbeite weiter an meinen LibTom-Projekten und bin auch immer bemüht, sie bekannt zu machen. Dazu nehme ich regelmäßig an Konferenzen wie Toorcon teil, um die LibTom-Philosophie zu verbreiten und hoffentlich weitere Open-Source-Entwickler auf den didaktischen Pfad zu bringen.

Simon Johnson ist ein Computerprogrammierer aus England, der als Sicherheitsingenieur mit C#-Anwendungen und Ähnlichem arbeitet und viel über Computersicherheit und Kryptografietechniken liest. Wir haben uns in der Usenet-Wüste von *sci.crypt* getroffen und bereits an verschiedenen Projekten zusammengearbeitet. Dieses Buch hat Simon als Fachgutachter betreut. Sein Terminplan hat ihm nicht immer erlaubt, so viel Zeit für dieses Projekt einzuräumen, wie er gern gehabt hätte, aber seine Hilfe war trotzdem von entscheidender Bedeutung. In den folgenden Jahren können wir außerdem ein oder zwei Bücher von Simon selbst erwarten.

Worum geht es nun in diesem Buch? Um Kryptografie für Entwickler. Das klingt sehr bestimmt, ganz nach »richtig« oder »falsch«. Dieses Buch ist im Grunde genommen ein Leitfaden für Entwickler, die selbst keine Kryptografen sind. Es sollte jedoch nicht als die alleinige Quelle zu diesem Thema angesehen werden. Wir verweisen selbst oft auf andere Bücher. Sie sollten sich auf jeden Fall auch ein Exemplar von *BigNum Math* zulegen, ein unverzichtbares Werk über Langzahlarithmetik, wie sie für Algorithmen mit öffentlichen Schlüsseln erforderlich ist. Ebenfalls unverzichtbar ist *The Guide to Elliptic Curve Cryptography* (ISBN 038795273X), das auf Einsteigerniveau alles abdeckt, was Entwickler

über Algorithmen mit elliptischen Kurven wissen müssen. Wir glauben, dass Sie viel mehr davon haben, wenn wir auf diese fundierten Texte hinweisen, anstatt ihre Aussagen hier noch einmal selbst zu formulieren. Es gibt auch einige Standards, die Sie sich ansehen sollten. Beispielsweise brauchen Sie unbedingt eine Ausfertigung von PKCS1 (die kostenlos erhältlich ist), wenn Sie RSA implementieren wollen. Dieses Buch behandelt zwar PKCS1-Operationen, doch es ist immer nützlich, den Standard selbst zur Hand zu haben. Abschließend möchte ich Ihnen noch dringend ans Herz legen, sich die LibTom-Projekte zu beschaffen, um praktische Erfahrungen mit Kryptografie-Software zu machen.

Für wen ist dieses Buch gedacht? Ich habe es für diejenigen Personen geschrieben, die mir E-Mails mit Bitten um Hilfe bei meinen Projekten schicken. In diesem Buch geht es jedoch nicht um diese Projekte, sondern um die Probleme, die Benutzer haben, wenn sie sie verwenden. Sehr oft sind die Entwickler, denen die Lösung von Sicherheitsproblemen übertragen wird, keine Kryptografen. Aber es sind kluge Köpfe, die mit einer entsprechenden Anleitung in der Lage sind, sichere Kryptosysteme einzurichten. Dieses Buch soll Entwicklern bei der Lösung kryptografischer Probleme helfen. Wenn Sie jemals vor der Frage standen, wie Sie AES bloß einrichten sollen, dann ist dies die richtige Lektüre für Sie.

Das Buch ist jedoch *nicht* für Menschen gedacht, die eine umfassende akademische Abhandlung über Kryptografie suchen. Es heißt nicht »Handbuch der angewandten Kryptografie« oder »Grundlagen der Kryptografie«. Einfach ausgedrückt: Wenn Sie nicht die Aufgabe haben, ein Kryptosystem zu implementieren, dann ist dieses Buch wahrscheinlich nicht das richtige für Sie. Diese Überlegung ist in die Gestaltung des Textes eingeflossen. Wir versuchen zwar, genügend technische und theoretische Einzelheiten für eine möglichst genaue und nutzbringende Darstellung zu geben, doch wir haben eine ganze Menge kryptografischer Details weggelassen, die von dem eigentlichen Zweck dieses Buch wegführen würden.

Ich möchte mehreren Personen für die Hilfe bei diesem Projekt danken. Greg Rose hat bei der Durchsicht eines Kapitels geholfen und auch einige Anregungen und erhellende Kommentare gegeben. Simon möchte ich für seine Mitarbeit und dafür danken, dass er zur Qualität dieses Textes beigetreten hat. Ich danke Microsoft Word dafür, es mir schwer gemacht zu haben. Des Weiteren möchte ich Andrew, Erin und den anderen bei Syngress dafür danken, dass sie dieses Buch realisiert haben. Mein Dank gilt auch den Benutzern des LibTom-Projekts, die die Inspiration zu diesem Buch waren. Ohne ihre Anfragen und ihre Erfahrungsberichte hätte ich gar kein Thema gehabt, über das ich schreiben könnte.

Abschließend möchte ich auch den Vorbestellern danken, die den verschobenen Drucktermin hingenommen haben. Das war meine Schuld.

Tom St Denis

Ottawa

Oktober 2006

Danksagung

Syngress möchte den folgenden Personen für ihre Freundlichkeit und ihre Unterstützung bei der Realisierung dieses Buchs danken:

Syngress-Bücher werden jetzt in den USA und in Kanada von O'Reilly Media vertrieben. Der Eifer und die Arbeitsmoral bei O'Reilly sind außerordentlich, und wir möchten allen für ihre Zeit und ihre Bemühungen danken, Syngress-Bücher auf den Markt zu bringen: Tim O'Reilly, Laura Baldwin, Mark Brokering, Mike Leonard, Donna Selenko, Bonnie Sheehan, Cindy Davis, Grant Kikkert, Opol Matsutaro, Steve Hazelwood, Mark Wilson, Rick Brown,Tim Hinton, Kyle Hart, Sara Winge, Peter Pardo, Leslie Crandell, Regina Aggio Wilkinson, Pascal Honscher, Preston Paull, Susan Thompson, Bruce Stewart, Laura Schmier, Sue Willing, Mark Jacobsen, Betsy Waliszewski, Kathryn Barrett, John Chodacki, Rob Bullington, Kerry Beck, Karen Montgomery und Patrick Dirden.

Wir danken den sehr hart arbeitenden Mitarbeitern bei Elsevier Science, die dafür sorgen, dass unsere Vorhaben ihren weltweiten Umfang behalten: Jonathan Bunkell, Ian Seager, Duncan Enright, David Burton, Rosanna Ramacciotti, Robert Fairbrother, Miguel Sanchez, Klaus Beran, Emma Wyatt, Krista Leppiko,Marcel Koppes, Judy Chappell, Radek Janousek, Rosie Moss, David Lockley, Nicola Haden, Bill Kennedy,Martina Morris, Kai Wuerfl-Davidek, Christiane Leipersberger, Yvonne Grueneklee, Nadia Balavoine und Chris Reinders.

Wir danken David Buckland, Marie Chieng, Lucy Chong, Leslie Lim, Audrey Gan, Pang Ai Hua, Joseph Chan, June Lim und Siti Zuraidah Ahmad von Pansing Distributors für die Begeisterung, mit der sie unsere Bücher aufnehmen.

Wir danken David Scott, Tricia Wilden, Marilla Burgess, Annette Scott, Andrew Swaffer, Stephen O'Donoghue, Bec Lowe,Mark Langley und Anyo Geddes von Woodslane für den Vertrieb unserer Bücher in Australien, Neuseeland, Papua-Neuguinea, Fidschi, Tonga, den Solomonen und den Cook-Inseln.

Der Autor

Tom St Denis ist ein Softwareentwickler, der vor allem für seine gemeinfreien Kryptografiebibliotheken aus der LibTom-Reihe bekannt ist. In den letzten Jahren hat er die Idee von Open-Source-Kryptografie entwickelt, weiterverbreitet und unterstützt und ist für ihre sichere Bereitstellung eingetreten. Zurzeit arbeitet er für Elliptic Semiconductor Inc., wo er Softwarebibliotheken für Embedded Systems entwirft und entwickelt. Dabei arbeitet er eng mit einem Team von Hardware-Ingenieuren zusammen, um eine bestmögliche Kombination aus Hardware und Software zu erreichen.

Zusammen mit Greg Rose ist Tom der Autor von *BigNum Math: Implementing Cryptographic Multiple Precisison Arithmetic* (Syngress Publishing, ISBN 1-59749-112-8), das Langzahlarithmetik für kryptografische Anwendungen erklärt.

Fachgutachter und Co-Autor

Simon Johnson ist Sicherheitsingenieur einer britischen Firma. Schon während seiner Teenagerzeit begann er sich für Kryptografie zu interessieren und studierte alle Aspekte konventioneller Software-Kryptografie. Seit dem Alter von 17 Jahren ist er aktives Mitglied der Usenet-Gruppe *Sci.Crypt*. Er nimmt an verschiedenen Sicherheitskonferenzen in aller Welt teil und setzt sich offen für sichere IT-Praktiken ein.

Einführung

1.1 Einleitung

Computersicherheit ist ein wichtiges Studiengebiet, dessen Ergebnisse sich auf die meisten alltäglichen Transaktionen auswirken. Sie kommt zum Tragen, wenn wir unsere Mobiltelefone einschalten, unsere Telefon-Mailbox abhören, unseren E-Mail-Posteingang einsehen, eine Debit- oder Kreditkarte verwenden, einen Per-per-view-Film bestellen, ein elektronisches Mautsystem nutzen und uns für Onlinespiele registrieren, ja, sogar schon, wenn wir unseren Hausarzt aufsuchen. Auch bei der Einrichtung virtueller privater Netzwerke (VPNs) und SSH-Verbindungen (Secure Shell), über die Angestellte mit ihrem Unternehmen kommunizieren und auf dessen Computer zugreifen können, spielt die Sicherheit eine große Rolle.

Hinter dem Gebrauch – und häufig auch dem Missbrauch – der Kryptografie zur Lösung von Sicherheitsproblemen steht vor allem ein Grund, nämlich das starke Bedürfnis nach Sicherheit. Sicherheit kommt aber nicht einfach dadurch zustande, dass man sie braucht, doch das wird leider oft erst im Nachhinein klar, nachdem es bereits einen Einbruch gegeben hat.

Berichte aus dem Untergrund
Ein klassischer Exploit: Dark Age of Camelot

Im März 2004 nutzte ein Exploit für das Onlinespiel *Dark Age of Camelot* (Mythic Entertainment) die schwache Serverauthentifizierung aus, die für die Durchführung der Abrechnungstransaktionen verwendet wurde. Dadurch konnten die Angreifer die Kommunikation zwischen einem echten Server und dem Client abhören und alle privaten Rechnungsdaten lesen.

Die Entwickler hatten zwar für den Kernalgorithmus eine bewährte und geprüfte kryptografische Bibliothek eingesetzt, den Algorithmus aber falsch angewendet. Dadurch mussten die Angreifer keinen schwierigen kryptografischen Algorithmus wie RSA oder RC4 knacken, sondern nur die schwache Konstruktion, in der er benutzt wurde.

Der Mythic-Exploit ist ein klassisches Beispiel für einen falschen Einsatz der richtigen Werkzeuge. Dabei wäre es jedoch unfair, dem Entwicklungsteam die Schuld zu geben. Schließlich handelt es sich um Spieleentwickler und nicht um Vollzeitkryptografen. Ihnen stand nicht das Budget zur Verfügung, um einen Krytografieexperten in ihr Team aufzunehmen oder gar die entsprechenden Aufgaben an eine andere Firma zu vergeben.

Die Lage, in der sich Mythic damals befand, war typisch für die meisten Softwareentwicklungsunternehmen in aller Welt. Da mehr und mehr kleine Firmen auf den Markt drängten, standen ihnen immer weniger Ressourcen für die Sicherheit zur Verfügung. Schließlich ist Sicherheit nicht das Ziel bei der Entwicklung eines Endbenutzerprodukts, sondern lediglich ein Erfordernis, damit dieses Produkt brauchbar ist.

Beispielsweise funktionieren Bankgeschäfte auch ganz ohne Kryptografie. Sie können jemandem einfach 10 € aushändigen, ohne erst einen RSA-Schlüsselaustausch durchführen zu müssen. Auch für das Funktionieren von Mobiltelefonen ist keine Kryptografie erforderlich. Die Digitalisierung des gesprochenen Wortes, die Komprimierung, die Codierung für die Übertragung über Funk und der umgekehrte Vorgang laufen ab, ohne dass wir auch nur einen Gedanken an Kryptografie verschwenden.

Da Sicherheit kein integraler Produktwert ist, wird sie entweder ganz vernachlässigt oder auf die Liste sekundärer, »wünschenswerter« Ziele abgeschoben. Das ist bedauerlich, insbesondere da sich die Kryptografie und ihre Bereitstellung oft leicht handhaben lassen und gar keine fortgeschrittenen Kenntnisse in Kryptografie oder Mathematik erfordern. Tatsächlich reicht es einfach aus zu wissen, *wie* die verfügbaren kryptografischen Werkzeuge für die anstehende Aufgabe eingesetzt werden müssen.

1.2 Bedrohungsmodelle

Ein Bedrohungsmodell stellt ausdrücklich die Angriffswege dar, die Ihre Gegner ausnutzen werden, um Ihr System um unterlaufen. Beim Angriff auf eine Bank geht es den Angreifern darum, Anmeldeinformationen zu erlangen, bei einem E-Mail-Dienstleister darum, private Nachrichten abzufangen. Einfach ausgedrückt, sehen wir uns bei einem Bedrohungsmodell das an, was in den Ausnahmefällen jenseits der normalen Benutzungen passiert. Was geschieht, wenn Sie eine Antwort aus der Menge X erwarten und man Ihnen stattdessen eine Antwort Y schickt, die nicht zu X gehört?

Das einfachste Beispiel dafür ist die Funktion `atoi()` in C, die häufig in Programmen verwendet wird, ohne an Fehlererkennung zu denken. Das Programm erwartet einen ASCII-codierten Integer. Was aber geschieht, wenn die Eingabe kein Integer ist? Dies ist nicht gerade eine Sicherheitslücke, aber genau die Art von Ausnahmefall, die Angreifer gern ausnutzen.

Ausgangspunkt für ein Bedrohungsmodell sind die Ebenen, auf denen alle – einschließlich Insider – mit dem System in Wechselwirkung treten können. Häufig werden Insider als besondere Benutzer behandelt. Da sie praktisch uneingeschränkten Zugriff auf die Daten haben, sind sie auch in der Lage, groteske Fehler zu begehen, z. B. die vertraulichen Daten von Tausenden von Benutzern in einem Laptop in einem Auto zurückzulassen.

Das Modell gibt im Grunde genommen an, was die *Usecases* des Systems erlauben, wenn sie missbraucht oder umgangen werden. Wenn beispielsweise ein Benutzer als Erstes sein Passwort angeben muss, können die Angreifer erkennen, wenn da Passwort nicht ordnungsgemäß gehandhabt wird. Sie können erkennen, ob das System Benutzer daran hindert, leicht zu erratende Passwörter auszuwählen usw.

Um ein präzises Bedrohungsmodell aufzustellen, ist es von entscheidender Bedeutung, sich die Usecases nicht im Hinblick darauf anzusehen, was ein normaler Benutzer tun wird. Wenn Ihr Programm beispielsweise Daten an eine Datenbank überträgt, können Angreifer einen *Injektionsangriff* durchführen, indem sie im Rahmen der eingegebenen Daten SQL-Abfragen übermitteln. Ein normaler Benutzer würde gar nicht auf diese Idee kommen. Es ist praktisch unmöglich, den kompletten Prozess zur Gestaltung eines Bedrohungsmodells zu dokumentieren, da jedes Bedrohungsmodell für ein System mindestens so kompliziert ist wie das System selbst.

Ich will Ihnen nicht verhehlen, dass dieses Buch das Schreiben von sicherem Code nicht in ausreichendem Maße behandelt, um dieses Problem zu lösen. Allerdings gibt es drei einfache Regeln für die Gestaltung von Bedrohungsmodellen, die Entwickler bei der Gestaltung ihres Systems beachten sollten:

1.2.1 Einfache Regeln zur Gestaltung des Bedrohungsmodells

1. Auf wie viele verschiedene Weisen kann jemand den vorliegenden Usecase übergehen?
   a. Denken Sie außerhalb der vorgesehenen Übergänge!
   b. Werden ungültige Kontexte behandelt?
2. Mit welchen Komponenten tritt die Eingabe in Interaktion?
   a. Was wären »ungültige Eingaben«?
3. Ist dieser Usecase effektiv?
   a. Ist er ausdrücklich schwach? Was sind die Annahmen?
   b. Erfüllt er das vorgesehene Ziel?

1.3 Was ist Kryptografie?

Kryptografie ist eine automatisierte (oder algorithmische) Methode, um Sicherheitsziele zu erreichen. Wenn wir von einem »Kryptoalgorithmus« sprechen, meinen wir damit gewöhnlich einen Algorithmus, der auf einem Computer ausgeführt werden soll. Diese Algorithmen bearbeiten Nachrichten aus Gruppen von Bits.

Bei dem Wort »Kryptografie« denken die meisten Leute an das Studieren von Chiffren, also an Algorithmen, die die Bedeutung einer Nachricht verschleiern. Das Ziel dabei, die Vertraulichkeit, ist jedoch nur eines von mehreren Zielen, das die Kryptografie erreichen soll. Es ist wahrscheinlich das am weitesten verbreitete und älteste Sicherheitsziel der Kryptografie, da es unserem natürlichen Bedürfnis entgegenkommt, Geheimnisse zu haben. Geheimnisse, etwa in Form von Wünschen, Begehren, Fehlern und Ängsten, sind natürliche Gefühle und Gedanken, die alle Menschen haben, was von Hollywood durch Filme wie *Passwort: Swordfish* und *Das Mercury-Puzzle* noch gefördert wird.

1.3.1 Ziele der Kryptografie

Es gibt jedoch noch andere Probleme, die die Kryptografie lösen soll, und sie können genauso wichtig oder sogar noch wichtiger sein – je nachdem, wer Sie angreift und was Sie gegen diese Angreifer zu schützen versuchen. Die Ziele der Kryptografie, die wir uns in diesem Buch ansehen werden, sind (in der Reihenfolge des Auftretens) Vertraulichkeit, Integrität, Authentifizierung und Nachweisbarkeit.

Vertraulichkeit

Vertraulichkeit heißt, dass die Bedeutung oder der Zweck einer Nachricht verborgen bleiben, insbesondere vor weiteren Beteiligten in einem Medium zur Informationsübertragung wie dem Internet, einem WLAN, einem Mobilfunknetz usw.

Erreicht wird Vertraulichkeit gewöhnlich durch *Chiffren mit symmetrischen Schlüsseln*. Diese Algorithmen nehmen einen Geheimschlüssel entgegen und verschlüsseln dann die ursprüngliche Nachricht – den *Klartext* – zum sogenannten *Chiffretext*. Der Chiffretext weist die gleiche Menge Entropie (Unsicherheit, oder einfach gesagt: Informationen) auf wie der Klartext. Dadurch muss der Empfänger nur den gleichen geheimen Schlüssel anwenden, um den ursprünglichen Klartext aus dem Chiffretext zurückzugewinnen.

Chiffren können in zwei verschiedenen Formen aufgestellt werden, die beide ihre Stärken und Schwächen aufweisen. In diesem Buch beschäftigen wir uns nur mit den *Blockchiffren* ausführlich, und zwar insbesondere mit dem Advanced Encryption Standard (AES) des NIST (National Institute for Standards and Technology). AES ist weit verbreitet, da er sowohl auf großen als auch kleinen Prozessoren und auf kostengünstiger Hardware angemessen läuft. Aufgrund ihrer universellen Einsetzbarkeit sind Blockchiffren auch allgemein weiter verbreitet als Stromchiffren. Wie wir noch sehen werden, lässt sich AES nutzen, um verschiedene Algorithmen für die Vertraulichkeit aufzustellen (darunter auch einen Modus, der einer Stromchiffre ähnelt), aber auch Algorithmen für Integrität und Authentifizierung. AES ist nicht proprietär, er ist gut dokumentiert und stützt sich auf solide kryptografische Theorien (siehe Abb. 1.1).

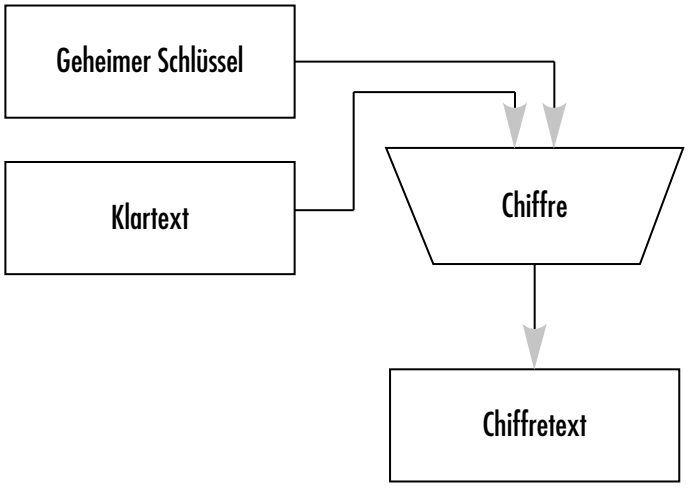

Abbildung 1.1: Blockdiagramm einer Blockchiffre

Hinweis
Der Advanced Encryption Standard ist die offizielle, vom NIST empfohlene Blockchiffre. Sie ist weit verbreitet und der direkte Nachfolger des veralteten und weit langsameren Data Encryption Standard (DES).

Integrität

Integrität ist die nachgewiesene Unversehrtheit in Abwesenheit eines aktiv vorgehenden Gegners. Das klingt komplizierter, als es in Wirklichkeit ist. Gemeint ist damit, dafür zu sorgen, dass eine Nachricht von Punkt A zu Punkt B übertragen werden kann, ohne dass sich die Bedeutung (oder der Inhalt) dabei ändert. Dabei betrachten wir nur Fälle, in denen kein Gegner aktiv versucht, die Richtigkeit der Übertragung zu beeinträchtigen.

Erreicht wird die Integrität gewöhnlich durch kryptografische *Einweg-Hashfunktionen*. Diese Funktionen nehmen als Eingabe eine Nachricht beliebiger Länge an und geben einen *Nachrichten-Digest* oder kurz *Digest* fester Länge zurück (gewöhnlich zwischen 160 und 512 Bits), der für die Nachricht steht. Anhand der Nachricht und des zugehörigen Digests lässt sich überprüfen, ob die Nachricht intakt übertragen wurde (unter der Voraussetzung, dass sie nicht von einem Angreifer aktiv manipuliert wurde). Hashalgorithmen (siehe Abb. 1.2) werden als kollisionsresistente Einwegfunktionen gestaltet.

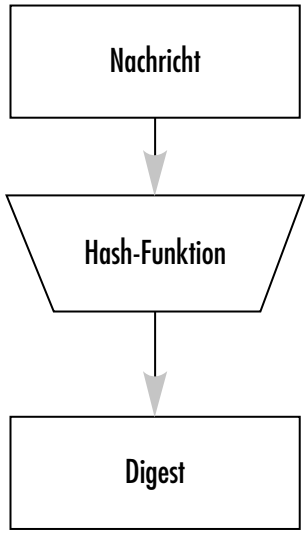

Abbildung 1.2: Blockdiagramm einer Einweg-Hashfunktion

Hashes werden als Einwegfunktionen gestaltet, was hauptsächlich darin liegt, dass sie als Methoden für die passwortgestützte Authentifizierung herangezogen werden. Es darf nicht möglich sein, die ursprüngliche Eingabe aus einem gegebenen Digest in einer praktikablen (d. h. weniger als exponentiellen) Zeit zu berechnen. Diese Einwegeigenschaft ist auch erforderlich, damit verschiedene darauf aufbauende Algorithmen, z. B. HMAC (Hash Message Authentication Code; siehe Kapitel 5) sicher gestaltet werden können.

Darüber hinaus müssen Hashes zwei bedeutende Formen der Kollisionsresistenz aufweisen. Erstens müssen sie Urbild- oder Preimage-Angriffen gegen ein festes Ziel widerstehen können (siehe Abb 1.3). Das bedeutet, bei gegebenem Wert *y* muss es schwer sein, eine Nachricht *M* zu finden, sodass *hash(M)= y*. Bei der zweiten geforderten Art der Resistenz – der Resistenz gegen einen Zweites-Urbild-Angriff (siehe Abb. 1.4) – ist es unmöglich, zwei Nachrichten *M1* (gegeben) und *M2* (willkürlich gewählt) zu finden, sodass *hash(M1) = hash(M2)*. Diese beiden Eigenschaften zusammengenommen bilden die Kollisionsresistenz.

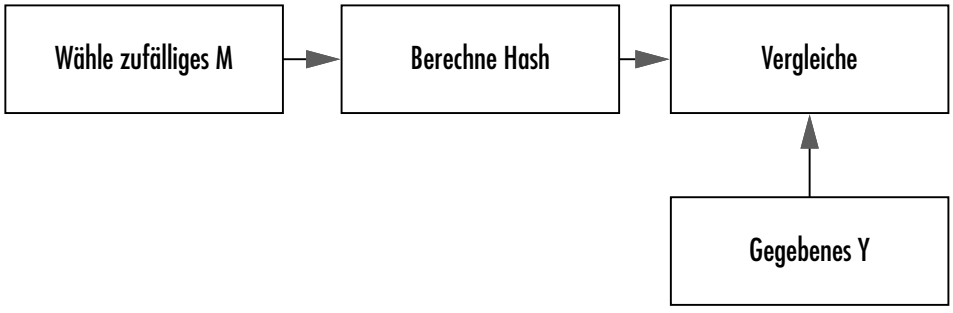

Abbildung 1.3: Resistenz gegen Urbild-Angriffe

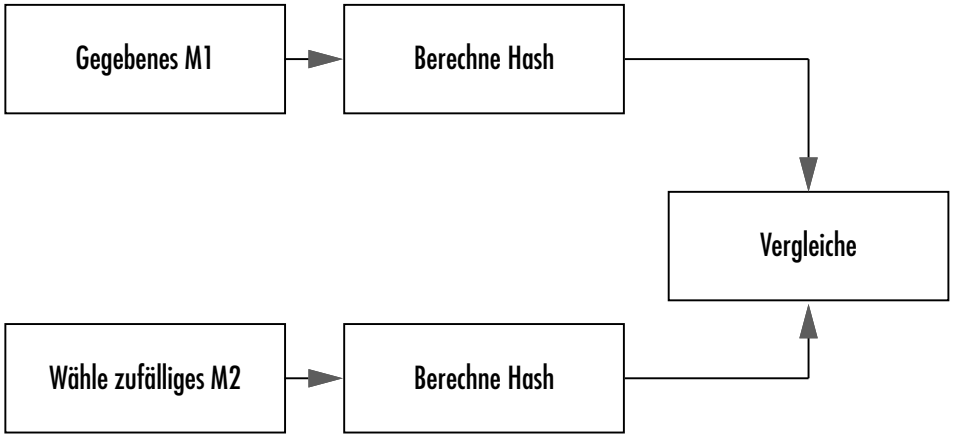

Abbildung 1.4: Resistenz gegen Zweites-Urbild-Angriffe

Hashalgorithmen verwenden keine Schlüssel. Es gibt also keine geheimen Informationen, die Angreifern für die Ausführung des Algorithmus fehlen. Wenn Sie den Digest einer öffentlich übermittelten Nachricht berechnen können, so können das auch die Angreifer. Daher kann die Integrität einer Nachricht damit nicht bestimmt werden, wenn ein Angreifer ins Spiel kommt.

Trotz dieser Einschränkung werden Hashes in der Informatik immer noch häufig eingesetzt. Beispielsweise wird bei den meisten Linux- und BSD-Distributionen im Rahmen des Dateimanifests auch ein Digest aus Programmen wie md5sum bereitgestellt. Bei einer Aktualisierung laden die Benutzer sowohl die erforderliche Datei (Tarball, RPM, DEB usw.) sowie deren Digest herunter. Das Bedrohungsmodell deckt dabei jedoch keine Angriffe ab, sondern lediglich Beschädigungen bei der Speicherung und Verbreitung wie verkürzte oder überschriebene Dateien.

In diesem Buch besprechen wir SHA-1 und SHA-2, zwei häufig verwendete Familien von Hashfunktionen aus dem Satz SHS (Secure Standard Hash) des NIST. Insbesondere die Familie SHA-2 ist sehr attraktiv, da sie eine Reihe von Hashes einschließt, die Digests einer Größe von 224 bis 512 Bit hervorrufen. Da sie keine Tabellen oder komplizierten Anweisungen benötigen, sind diese Algorithmen auch ziemlich effizient. Außerdem lassen sie sich von der Spezifikation ausgehend relativ leicht reproduzieren.

Warnung
Der Hashalgorithmus MD5 wird schon seit langer Zeit als ziemlich schwach eingestuft. Dobbertin hat Mängel in den Kernkomponenten des Algorithmus gefunden, und 2005 entdeckten Forscher Kollisionen bei den Funktionen. Arbeiten von Anfang 2006 zeigten immer schnellere Methoden zum Auffinden von Kollisionen auf.

Dabei suchen die Forscher hauptsächlich nach Zweites-Urbild-Kollisionen. Es gibt bereits Methoden, diese Kollisionen gegen IDS und verteilte Systeme einzusetzen.

Entwicklern wird dringend davon abgeraten, die Hashfunktion MD5 zu verwenden. Selbst auf SHA-1 sollte zugunsten von SHA-2 verzichtet werden. Für Entwickler, die die europäischen Standards befolgen, steht auch die Whirlpool-Hashfunktion zur Verfügung.

Authentifizierung

Authentifizierung bedeutet, eine Identität oder Repräsentation der Integrität einer Nachricht bereitzustellen. Ein klassisches Beispiel dafür ist ein Wachssiegel auf einem Brief. Diese Kennzeichnungen ließen sich zu der Zeit, als sie in Gebrauch waren, nur schwer

fälschen, weshalb das Vorhandensein eines unbeschädigten Siegels bedeutete, dass die enthaltenen Dokumente authentisch waren.

Eine andere gebräuchliche Form der Authentifizierung ist die Eingabe einer persönlichen Identifizierungsnummer (PIN) oder eines Passworts, um eine Transaktion zu autorisieren. Das ist nicht mit der Nachweisbarkeit zu verwechseln, also der Unmöglichkeit, eine Vereinbarung anzufechten, oder mit Authentifizierungsprotokollen, soweit es die Aushandlung einer Einrichtung von Schlüsseln betrifft. Wenn wir davon sprechen, eine Nachricht zu authentifizieren, meinen wir damit, zusätzliche Schritte auszuführen, sodass der Empfänger die Integrität einer Nachricht auch in Anwesenheit eines aktiven Angreifers sicherstellen kann.

Die Aushandlung von Schlüsseln und das verwandte Gebiet der Authentizität gehören in den Bereich der Protokolle mit öffentlichen Schlüsseln. Sie verwenden größtenteils die gleichen Grundelemente, weisen aber andere Einschränkungen und Ziele auf. Ein Authentifizierungsalgorithmus soll gewöhnlich symmetrisch sein, sodass alle Parteien verifizierbare Daten produzieren können. Nachweisbare Authentizität dagegen ist gewöhnlich Sache eines einzelnen Produzenten und vieler Verifizierer.

In der Kryptografie werden diese Authentifizierungsalgorithmen auch oft als MAC (Message Authentication Codes) bezeichnet. Ebenso wie Hashfunktionen weist ihre Ausgabe, das sogenannte Nachrichten-*Tag*, eine feste Größe auf. Dieses Tag ist die Information, die ein Verifizierer benötigt, um ein Dokument zu validieren. Anders als bei Hashfunktionen ist für MAC-Funktionen ein geheimer Schlüssel erforderlich, um zu verhindern, dass Tags gefälscht werden (siehe Abb. 1.5).

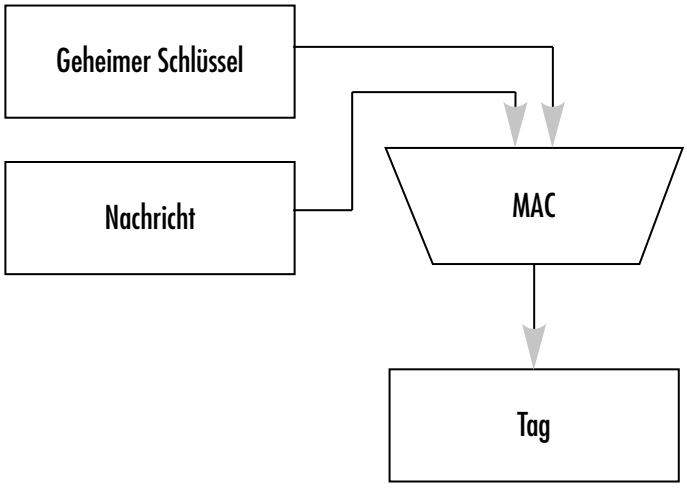

Abbildung 1.5: Blockdiagramm einer MAC-Funktion

Die beiden gebräuchlichsten Formen von MAC-Algorithmen sind CBC-MAC (inzwischen durch den Algorithmus OMAC1 implementiert und in den Kreisen des NIST als

CMAC bezeichnet) und HMAC. Die CBC-MAC-Funktionen (oder CMAC-Funktionen) verwenden eine Blockchiffre, die HMAC-Funktionen eine Hashfunktion. Dieses Buch behandelt beide Formen.

Eine weitere Methode zur Authentifizierung sind Algorithmen mit öffentlichen Schlüsseln wie RSA mit dem Standard PKCS1 oder EC-DSA (Elliptic Curve DSA oder ANSI X9.62). Anders als bei CMAC oder HMAC ist es bei einem Authentifizierer auf der Grundlage öffentlicher Schlüssel nicht erforderlich, dass die beiden Parteien vor der Kommunikation private Informationen austauschen. Daher sind Algorithmen mit öffentlichen Schlüsseln bei der Kommunikation mit willkürlichen, unbekannten Parteien nicht auf Onlinetransaktionen beschränkt. Das bedeutet, dass Sie ein Dokument signieren können, das dann jeder mit Zugriff auf Ihren öffentlichen Schlüssel verifizieren kann, ohne erst mit Ihnen Rücksprache zu halten.

Algorithmen mit öffentlichen Schlüsseln werden ganz anderes eingesetzt als MAC-Algorithmen. Daher werden sie in diesem Buch weiter hinten als eigenes Thema behandelt.

Tabelle 1.1: Vergleich von CMAC, HMAC und Algorithmen mit öffentlichem Schlüssel

Merkmal	CMAC	HMAC	RSA PKCS 1	EC-DSA
Geschwindigkeit	Schnell	Am schnellsten	Am langsamsten	Langsam
Komplexität	Einfach	Am einfachsten	Schwer	Am schwersten
Geheimer Schlüssel erforderlich	Ja	Ja	Nein	Nein
Bereitstellung	Nicht so einfach	Nicht so einfach	Einfach	Einfach

Diese verschiedenen Arten von Authentifizierern werden unterschiedlich eingesetzt. Authentifizierer auf der Grundlage öffentlicher Schlüssel dienen gewöhnlich dazu, eine erste Begrüßung über ein neu eröffnetes Kommunikationsmedium auszuhandeln. Beispielsweise verifizieren Sie bei der ersten Verbindung zu einer SSL-Website, dass die Signatur des Zertifikats dieser Website tatsächlich von einer Stammzertifizierungsstelle stammt. Dagegen werden CMAC- und HMAC-Algorithmen gewöhnlich erst nach der Absicherung des Kommunikationskanals eingesetzt, um sich zu vergewissern, dass der Datenverkehr übertragen wurde, ohne dass sich jemand daran zu schaffen gemacht hat. Da CMAC und HMAC Nachrichtendaten viel schneller verarbeiten können, eignen sie sich viel besser für Medien mit hohem Verkehrsaufkommen.

Nachweisbarkeit

Nachweisbarkeit bedeutet, dass es nicht möglich ist, eine eingegangene Verpflichtung zu widerrufen. Wenn Sie beispielsweise einen Vertrag mit einem Stift unterzeichnen, dient Ihre Unterschrift als Nachweis, dass Sie den Vertrag eingegangen sind. Sie können später

nicht einfach den Vertragsbedingungen widersprechen oder gar behaupten, diese Übereinkunft nicht eingegangen zu sein.

Nachweisbarkeit und Authentifizierung ähneln sich in der Hinsicht, dass für ihre Umsetzung oft die gleichen Grundelemente verwendet werden. Beispielsweise kann die Signatur eines öffentlichen Schlüssels als Nachweis dienen, wenn nur eine Partei die Fähigkeit hat, Signaturen durchzuführen. Daher können andere MAC-Algorithmen wie CMAC oder HMAC nicht zur Nachweisbarkeit herangezogen werden.

Die Nachweisbarkeit ist eine sehr wichtige Eigenschaft bei Rechnungsstellung und Buchung, wird aber oft nur unzureichend umgesetzt. Beispielsweise werden Unterschriften auf Kreditkartenquittungen nur selten überprüft, und selbst wenn jemand einen flüchtigen Kontrollblick auf die Rückseite der Karte wirft, so handelt es sich doch selten um einen Handschriftenexperten, der eine Fälschung erkennen könnte. In Mobiltelefonen werden gewöhnlich MAC-Algorithmen als Authentifizierer für die Ressourcennutzung verwendet. Dort haben diese Algorithmen daher keine Eignung für die Nachweisbarkeit.

Die Ziele im Überblick

Tabelle 1.2 gibt einen Überblick über die vier wichtigsten Ziele der Kryptografie.

Tabelle 1.2: Die wichtigsten Ziele der Kryptografie

Ziel	Eigenschaften
Vertraulichkeit	1. Betrifft die Verschleierung der Bedeutung einer Nachricht vor unbefugten anderen Teilnehmern in einem Kommunikationsmedium. 2. Gelöst durch Blockchiffren mit symmetrischen Schlüsseln. 3. Der Empfänger weiß nicht, ob die Nachricht intakt ist. 4. Die Ausgabe einer Chiffre ist ein Chiffretext.
Integrität	1. Betrifft die Intaktheit einer Nachricht bei der Übertragung. 2. Geht davon aus, dass keine gegnerischen Aktivitäten vorliegen. 3. Gelöst durch Einweg-Hashfunktionen. 4. Die Ausgabe eines Hashes ist ein Nachrichten-Digest.
Authentifizierung	1. Betrifft die Intaktheit einer Nachricht bei der Übertragung. 2. Geht davon aus, dass gegnerische Aktivitäten vorliegen. 3. Gelöst durch MAC-Funktionen (Message Authentication Code). 4. Die Ausgabe einer MAC-Funktion ist ein Nachrichten-Tag.
Nachweisbarkeit	1. Betrifft bindende Transaktionen. 2. Das Ziel besteht darin, eine Partei daran zu hindern, die Teilnahme an der Transaktion zu widerrufen. 3. Gelöst durch digitale Verschlüsselungen mit öffentlichem Schlüssel. 4. Die Ausgabe eines Signaturalgorithmus ist eine Signatur.

1.4 Handhabung der schützenswerten Güter

Eine große Herausforderung bei der Umsetzung der Kryptografie besteht darin, auf sichere und effiziente Weise mit den Benutzergütern und den Identitätsnachweisen umzugehen. Bei diesen »Gütern« handelt es sich um alles, was der Benutzer geschützt haben möchte, beispielsweise Nachrichten und Dateien, aber auch um medizinische Daten oder Kontaktlisten. Es handelt sich um Dinge, die der Benutzer besitzt, die ihn aber nicht eindeutig identifizieren; oder genauer gesagt, die normalerweise nicht dazu herangezogen werden, ihn eindeutig zu identifizieren. Die Identitätsnachweise dagegen tun genau das. Darunter fallen Dinge wie Benutzernamen, Passwörter, PINs, Token zur Zwei-Faktor-Authentifizierung, RFID-Plaketten usw. Auch Informationen wie private RSA- und ECC-Schlüssel für Signaturen können darunter fallen.

Der Umgang mit den Benutzergütern erfolgt im Großen und Ganzen nicht gerade in einer besonders sicheren Art und Weise. Sie werden gewöhnlich als authentisch angesehen, und ihre Vertraulichkeit wird nur selten geschützt. Nur wenige Programme b enthalten eingebaute Sicherheitsfunktionen für solche Benutzergüter. Stattdessen wird meistens davon ausgegangen, dass der Benutzer selbst zusätzliche Werkzeuge einsetzt, z. B. verschlüsselte Speicherorte oder GnuPG.

Manchmal werden Benutzergüter auch als Identitätsnachweis missbraucht. Beispielsweise war es 2005 möglich, mit einer Kreditkarte als »Ausweis« quer durch Kanada zu fliegen. E-Tickets konnten an automatischen Check-in-Terminals abgeholt werden, und beim Boarding hat das Personal bei Inlandsflügen nicht auf Fotos zur Identifizierung geachtet. Man ging davon aus, dass der Besitz einer Kreditkarte auch bedeutete, dass tatsächlich die Person, die das Ticket erworben hatte, am Gate stand.

1.4.1 Vertraulichkeit und Authentifizierung

Bei Benutzergütern müssen wir uns zwei wichtige Fragen stellen, nämlich erstens: »Sind sie privat?«, und zweitens: »Müssen sie intakt sein?« Beispielsweise wenden viele Benutzer von Tools zur Festplattenverschlüsselung dieses Werkzeug auf ihr gesamtes Systemlaufwerk an. Viele Systemdateien sind als Teil des Installationsmediums aber universell zugänglich. Es handelt also um alles andere als um private Güter, weshalb es eine Ressourcenverschwendung ist, sie zu verschlüsseln. Schlimmer ist jedoch, dass die meisten dieser Systeme keine Authentifizierungswerkzeuge auf die Dateien auf der Festplatte anwenden. (Eine rühmliche Ausnahme bildet EncFS; *http://encfs.sourceforge.net/*.) Das bedeutet, dass viele Dateien, einschließlich Anwendungen, die für den Benutzer zugänglich sind, verändert werden können. Das Schlimmste, was damit gewöhnlich erreicht werden kann, ist ein Denial-of-Service-Angriff (DoS) auf das System. Es ist jedoch durchaus möglich, ein Programm so zu verändern, dass dadurch Schwachstellen in das System eingeführt werden. Mit Authentifizierung ist eine Datei entweder lesbar oder nicht. Änderungen werden schlicht nicht akzeptiert.

Wenn Informationen nicht für die Öffentlichkeit gedacht sind, ist es im Allgemeinen eine gute Idee, sie zu authentifizieren. Dadurch erhalten die Benutzer einen doppelten Schutz

für ihre Güter. Angesichts dieses neuen Trend empfehlen NIST und IEEE einen kombinierten Modus (CCM bzw. GCM), der sowohl für Verschlüsselung als auch für Authentifizierung sorgt. Diese Modi sind auch von theoretischem Interesse, da sie formal auf die Sicherheit ihrer ererbten Primitiva (gewöhnlich der Blockchiffre AES) zurückgeführt werden können. Allerdings bieten diese Modi eine geringere Leistung als Verschlüsselung oder Authentifizierung allein. Das wird jedoch durch die Stabilität ausgeglichen, die sie dem Benutzer bieten.

1.4.2 Lebenszyklus von Daten

Während der Lebensdauer von Gütern oder Identitätsnachweisen kann es notwendig sein, Aktualisierungen oder Ergänzungen daran vorzunehmen oder sogar sie ganz oder teilweise zu entfernen. Anders als beim einfachen Erstellen eines neuen Guts sind die Konsequenzen für die Sicherheit beim Zulassen späterer Änderungen alles andere als trivial. Einzelne Betriebsmodi sind nicht sicher, wenn ihre Parameter unverändert bleiben. Beispielsweise erfordert die CTR-Verkettungsmethode (siehe Kapitel 4) bei jeder Datenverschlüsselung einen frischen Ausgangswert. Bei der einfachen Wiederverwendung eines vorhandenen Ausgangswerts, etwa um eine Inline-Veränderung zuzulassen, besteht keinerlei Schutz vor Bedrohungen der Vertraulichkeit. Andere Modi wie CCM (siehe Kapitel 7) erfordern einen frischen Nonce-Wert pro Nachricht, um sowohl die Vertraulichkeit als auch die Authentizität der Ausgabe zu gewährleisten.

Manche Daten, z. B. medizinische Daten, haben auch eine *Lebensdauer*, was, in kryptografischen Begriffen ausgedrückt, Einschränkungen bei der Zugriffssteuerung mit sich bringt. Gewöhnlich werden sie durch digitale Signaturen mit öffentlichem Schlüssel umgesetzt, die ein Ablaufdatum aufweisen. Streng genommen erfordert die Zugriffssteuerung einen vertrauenswürdigen Verteiler (z. B. einen Server), der die für die zu verteilenden Daten festgelegten Regeln (z. B. strikte Freiwilligkeit) einhält.

Das Prinzip von Datenströmen in einer bestimmten Reihenfolge entspricht der Notwendigkeit einer zustandsbehafteten Kommunikation. Wenn Sie beispielsweise beim Onlinebanking eine Anforderung an den Server senden, wollen Sie, dass die Anforderung nur einmal ausgegeben und akzeptiert wird – vor allem dann, wenn Sie eine Rechnung begleichen! Zu einem *Replay-Angriff* kommt es, wenn es einem Angreifer gelingt, Ereignisse in der Kommunikationssitzung erneut auszugeben, sodass sie erneut vom Empfänger akzeptiert werden. Die einfachste (und erste) Lösung besteht darin, Zeitstempel oder Inkrementzähler in die zu authentifizierenden Nachrichten aufzunehmen.

Allerdings sind Zeitstempel in der Kryptografie und ganz allgemein in der Informatik eine heikle Sache. Das fängt schon mit der Frage an, wie spät es eigentlich ist. Wahrscheinlich hat Ihr Computer eine andere Uhrzeit als meiner. Noch komplizierter wird es bei Onlineprotokollen, bei denen wir in Echtzeit mit Uhren zu tun haben können, die nicht nur abweichende Uhrzeiten zeigen, sondern auch unterschiedlich schnell laufen. Wenn Sie dazu noch die Natur von UDP-Netzwerken hinzunehmen, in denen die Reihenfolge der übertragenen Pakete nicht garantiert ist, kommen Sie ziemlich schnell zu ziemlich

monströsen Onlineprotokollen. Daher werden Zähler lieber gesehen als Timer. Doch so praktisch sie auch sein mögen, sind sie leider nicht überall anwendbar. Beispielsweise gibt es bei Offlineprotokollen so etwas wie Zähler überhaupt nicht, da für sie die Nachricht, die sie gerade sehen, immer die erste Nachricht ist. Es gibt dort keine »zweite« Nachricht.

Als gute Faustregel sollten Sie einen Zähler in den Teil des Datenkanals aufnehmen, der für die Authentifizierung zuständig ist. Vor allem aber müssen Sie diesen Zähler auch prüfen. Manchmal ist es besser, außerhalb der Reihenfolge eintreffende Nachrichten stillschweigend zu verwerfen, anstatt blind jeglichen Datenverkehr unabhängig von der Reihenfolge durchzulassen. In den Modi GCM und CCM gibt es Ausgangswerte bzw. Nonces, die sich auf einfache Weise sowohl als Ausgangswert als auch als Zähler nutzen lassen.

Tipp
Ein einfacher Trick für Ausgangswerte oder Nonces in Protokollen wie GCM und CCM besteht darin, einige Bytes als Teil eines Paketzählers zu verwenden. Beispielsweise akzeptiert CCM 13-Byte-Nonces, wenn das Paket weniger als 65.536 Klartextbytes umfasst. Wenn Sie wissen, dass Sie weniger als vier Milliarden Pakete haben, können Sie die ersten vier Bytes als Paketzähler verwenden. Mehr darüber erfahren Sie in Kapitel 7.

1.5 Irrige Annahmen

Unter Kryptografen kursiert die Behauptung, dass man Sicherheit am besten den Kryptografen überlässt und dass die öffentliche Erörterung kryptografischer Algorithmen nur zu deren falscher Anwendung führt. So schrieb Bruce Schneier in der Zusammenfassung seines Buchs *Secret and Lies*:

> *Ein Grund für mich, dieses Buch zu schreiben, bestand darin, einen Fehler zu korrigieren.*
>
> *Vor sieben Jahren habe ich das Buch Applied Cryptography verfasst und darin ein mathematisches Utopia beschrieben: Algorithmen, die unsere innersten Geheimnisse jahrtausendelang bewahren; Protokolle, die die fantastischsten elektronischen Interaktionen – nicht reguliertes Glücksspiel, Authentifizierung im Verborgenen, anonyme Barzahlung – gefahrlos und sicher ausführen können. In meiner Vision wurde die Kryptografie zu einem großartigen technischen Werkzeug, das soziale Unterschiede beseitigte: Wer auch nur über einen billigen Computer verfügt (die alljährlich billiger werden), kann sich eine ebensolche Sicherheit leisten wie die große staatliche Behörden. In der zweiten Ausgabe jenes Buches, ging ich zwei Jahre später sogar so weit zu sagen: »Es reicht nicht, dass wir uns mithilfe von Gesetzen schützen; wir müssen uns mithilfe der Mathematik schützen« (Bruce Schneider, Secrets and Lies)*

Hier sagt Schneier, dass die Kryptografie allein uns nicht »sicher« machen kann. Daran mag in der Praxis tatsächlich ein Körnchen Wahrheit sein. Es gibt viele Fälle, in denen eine an und für sich perfekte Kryptografie durch Benutzerfehler oder physische Angriffe (etwa Kartendiebstahl) ausgehebelt wurde. Der Gedanke jedoch, dass die Weitergabe kryptografischer Kenntnisse an Laien etwas wäre, für das man sich schämen müsste oder das man vermeiden sollte, ist jedoch schlicht falsch.

Wer sich einzig und allein auf Sicherheitsexperten verlässt, bekommt zwar garantiert ein sicheres System, aber oftmals ist eine solche Vorgehensweise unwirtschaftlich oder nicht praktikabel. Eines der Hauptanliegen dieses Buchs besteht darin, mit der Vorstellung aufzuräumen dass Kryptografie schwerer wäre (oder sein müsste), als sie in Wirklichkeit ist. Häufig kann ein klares Verständnis der Bedrohungen zu effizienten und leicht zu entwerfenden Kryptosystemen führen, die die vorliegenden Sicherheitsbedürfnisse erfüllen. Es gibt bereits viele anscheinend sichere Produkte, deren Entwickler nicht viel mehr getan haben, als nach den richtigen Werkzeugen für die vorliegende Aufgabe – Softwarebibliotheken, Algorithmen bzw. Implementierungen – und einer Methode zu suchen, sie auf sichere Weise anzuwenden.

Behalten Sie immer im Hinterkopf, worum es in diesem Buch geht: Wir verwenden Standardalgorithmen, die bereits von Kryptografen entwickelt werden, und sehen uns an, wie wir sie korrekt einsetzen können. Die Entwicklung und die Anwendung sind zwei sehr verschiedene Aufgaben. Für die erste sind sehr gute Kenntnisse in Kryptografie und Mathematik erforderlich. Für die zweite müssen wir lediglich wissen, welche Probleme die Algorithmen lösen sollen, warum sie so sind, wie sie sind, und wie wir sie nicht einsetzen dürfen.

Als wir vor kurzem das Entwicklungsteam für ein Onlinespiel besucht haben, haben wir uns das Kryptosystem angesehen, das unbemerkt von den Endbenutzern hinter den Kulissen läuft. Es fielen uns zwar sofort einige Dinge auf, die wir persönlich anders gestaltet hätten, aber nachdem wir die Sache einige Stunden lang untersucht hatten, konnten wir nichts erkennen, was auffällig falsch war. Die Entwickler hatten ein Bedrohungsmodell aufgestellt und etwas geschaffen, das mit der vorhandenen Rechenleistung machbar war und dieses Modell abdeckte. Kurz gesagt, sie hatten ein funktionierendes System gestaltet, das es ihnen erlaubte, ihr Produkt öffentlich bereitzustellen. Keines der Teammitglieder hatte Kryptografie studiert oder auch nur zu seinem Hobby gemacht.

Es stimmt zwar, dass es zu gefährlich unangemessenen Ergebnissen führen kann, wenn man nicht mehr weiß, als dass man Sicherheit braucht und dass es Algorithmen dafür gibt. Das ist ein Grund für dieses Buch: Es soll Ihnen zeigen, welche Algorithmen es gibt, wie Sie sie auf verschiedene Weisen einsetzen und welche Gefahren ihre Bereitstellung und Verwendung bietet. In einem Rezept können die Zutatenliste und die Anleitung zur Zubereitung auch in ein und demselben Text stehen.

1.6 Werkzeuge für Entwickler

Im Verlaufe dieses Buchs nutzen wir Tools wie GNU Compiler Collection C (GCC) für die x86-Plattform mit 32 oder 64 Bit sowie ARMv4-Prozessoren. Es handelt sich um professionelle, kostenlose und wertvolle Werkzeuge.

Die verschiedenen in diesem Buch vorgestellten Algorithmen sind größtenteils in portierbarem C geschrieben, damit sie möglichst weiträumig genutzt werden können. Bei den Listings handelt es sich um vollständige C-Routinen, die Sie sofort einsetzen können. An passender Stelle werden auch die vom C-Compiler generierten Assemblerlistings analysiert. Sie sind besonders geeignet für Leser, die mit den x86-Assemblern im Stil von AT&T oder den ARMv4-Assemblern vertraut sind.

Für die meisten Algorithmen oder Probleme sind mehrere Implementierungstechniken möglich. Beispielsweise gibt es für die Multiplikation drei zugrunde liegende praktische Algorithmen und dafür jeweils wieder verschiedene Implementierungen. Wo es sinnvoll ist, werden die Größe der einzelnen Konfigurationen und ihre Effizienz auf den genannten Plattformen verglichen. Vergleiche für die in diesem Buch nicht abgedeckten Plattformen sollten Sie selbst anstellen.

Bei manchen Algorithmen sind Abwägungen zu treffen. Beispielsweise ist AES am schnellsten, wenn Nachschlagetabellen verwendet werden, doch in einer solchen Konfiguration können *Nebenkanalinformationen* durchsickern (siehe Kapitel 4). In diesem Buch sehen wir uns Varianten an, um das Auftreten solcher Lecks zu minimieren. Die Analyse dieser Implementierungen ist eng mit dem Design moderner Prozessoren verknüpft, insbesondere mit den Datencaches. Sie sollen sich zumindest damit vertraut machen, wie x86-Prozessoren, etwa der Intel Pentium 4 oder der AMD Athol64 – auf Blockebene arbeiten.

Gelegentlich wird in diesem Buch auf bereits vorhandene Arbeiten wie TomsFastMath und LibTomCrypt verwiesen. Dabei handelt es sich um gemeinfreie Open-Source-Bibliotheken, die in C geschrieben wurden und überall in der Branche auf Plattformen unterschiedlichster Größe eingesetzt werden, von kleinen Netzwerksensoren bis zu großen Unternehmensservern. Die Codelistings in diesem Buch sind zwar unabhängig von diesen Bibliotheken, aber Sie sollten trotzdem deren Aufbau studieren und sie verwenden, anstatt das Rad neu zu erfinden. Das heißt natürlich nicht, dass Sie nicht auch versuchen sollten, die Algorithmen selbst zu implementieren, doch wenn die Umstände es erlauben, kann die Verwendung des verfügbaren Quellcodes die Produktentwicklung beschleunigen und die Testphase verkürzen. An Ihren eigenen Implementierungen sollen Sie sich dann versuchen, wenn es keine passenden Bibliotheken für die Bedingungen Ihres Projekts gibt. Alle Projekte stehen auf der Website *www.libtom.net* zur Verfügung.

Daten über die zeitliche Abstimmung auf x86-Prozessoren wurden mit der Anweisung RDTSC gewonnen, die eine zyklusgenaue Erfassung ermöglicht. Auch mit dieser Anweisung und ihrer Verwendung sollten Sie sich vertraut machen.

1.7 Zusammenfasung

Wie wir noch sehen werden, ist die Entwicklung professioneller Kryptosysteme keine sehr komplizierte Aufgabe ausschließlich für Experten. Durch die klare Definition eines Bedrohungsmodells, das die gefährdeten Punkte des Systems abdeckt, lässt sich erkennen, wie die Kryptografie zur Sicherheit eines Produkts beitragen kann. Kryptografie ist nur dann schwer fassbar, wenn man nicht weiß, wie man nach den Schwachstellen suchen soll.

Die Aufstellung eines Bedrohungsmodells aus allen Usecases kann zumindest teilweise als erledigt betrachtet werden, wenn die Fragen: »Deckt das die Vertraulichkeit ab?« und »Deckt das die Authentifizierung ab?« entweder mit »Ja!« oder mit »Das kommt hier nicht zum Tragen.« beantwortet werden.

Zu wissen, wo die Schwächen liegen, ist nur der erste Schritt zu einem sicheren Kryptosystem. Als Nächstes müssen Sie bestimmen, welche Lösungen Ihnen die Kryptografie für das Problem bietet – Blockchiffren, Hashes, Zufallszahlengeneratoren, Kryptografie mit öffentlichen Schlüsseln, MAC-Funktionen oder Challenge-Response-Systeme. Wenn Sie ein System entwerfen, dass einen öffentlichen RSA-Schlüssel benötigt, der Benutzer aber keinen hat, kann das System offensichtlich nicht funktionieren. Der Umgang mit Identitätsnachweisen und schützenswerten Gütern ist ein integraler Bestandteil des Designs von Kryptosystemen. Dieser Aspekt bestimmt, was Sie schützen müssen und welche kryptografischen Algorithmen geeignet sind.

Die Laufzeiteinschränkungen eines Produkts legen den verfügbaren Platz (Arbeitsspeicher) für den Programmcode und die Daten sowie die verfügbare Rechenleistung für eine gegebene Aufgabe fest. Sehr oft spielen diese Einschränkungen eine entscheidende Rolle bei der Auswahl der Algorithmen und des anschließenden Designs der in dem Kryptosystem verwendeten Protokolle. Wenn Sie an eine bestimmte CPU gebunden sind, kann ECC besser geeignet sein als RSA, und in manchen Fällen ist eine Verschlüsselung mit öffentlichen Schlüsseln nicht praktikabel.

Alle diese Designparameter – von der Natur des Programms über die schützenswerten Benutzergüter und Identitätsnachweise und schließlich die Laufzeitumgebung – müssen bei der Gestaltung eines Kryptosystems ständig abgestimmt werden. Diese Abstimmung und das Design einer geeigneten Lösung sind Aufgabe des Sicherheitsingenieurs. Dieses Buch hilft dabei, indem es das Was, Wie und Warum von kryptografischen Algorithmen beschreibt.

1.8 Der Aufbau dieses Buchs

Dieses Buch ist kapitelweise in Problemkategorien gegliedert. Jedes Kapitel stellt eine vollständige Beschreibung des zugehörigen Bereichs der Kryptografie dar, so weit er Entwickler betrifft. Dieses erste Kapitel dient dabei als Einführung in das Gebiet der Kryptografie im Allgemeinen. Wenn Sie eine ausführlichere Darstellung der Kryptografie haben möchten, sollten Sie Bücher wie *Angewandte Kryptographie*, *Practical Cryptography* oder *The Handbook of Applied Cryptography* lesen.

Kapitel 2 (»ASN.1-Codierung«) stellt die ASN.1-Codierungsregeln (Abstract Syntax Notation One) für Datenelemente wie Strings, Binärstrings, Integer, Datums- und Uhrzeitangeben, Mengen und Folgen vor. ASN.1 wird überall in Standards für öffentliche Schlüssel als Standardmethode für die Übertragung von Datenstrukturen mehrerer Typen in Mehrplattformumgebungen eingesetzt. Es ist auch sehr nützlich für allgemeine Datenstrukturen, da es sich um einen Standard und einen gut verstandenen Satz von Codierregeln handeln. Beispielsweise können Sie Dateiheader im ASN.1-Format codieren und Ihren Benutzern die Möglichkeit geben, Drittanbieterwerkzeuge einzusetzen, um die Header selbst zu debuggen und zu ändern. Mit der Verwendung einer ASN.1-Codierung anstelle eigener Standards erhält jedes Projekt einen erheblichen Mehrwert. In diesem Kapitel wird eine Teilmenge der ASN.1-Regeln eigens für gängige kryptografische Aufgaben behandelt.

Kapitel 3 (»Zufallszahlen«) beschreibt das Design und die Konstruktion von Standard-Zufallszahlengeneratoren (Random Number Generators, RNGs) wie denjenigen, die das NIST spezifiziert. RNGs und Pseudo- oder deterministische RNGs (PRNGs bzw. DRNGs) sind wichtige Bestandteile aller Kryptosysteme, da die meisten Algorithmen randomisiert sind und unvorhersehbare und nicht wiederholbare Eingaben (wie Anfangsvektoren oder Nonces) benötigen. Da in praktisch allen Kryptosystemen PRNGs vorhanden sind, beginnen wir unsere Besprechung mit ihnen. In diesem Kapitel werden die verschiedenen PRNG-Konstruktionen, ihre Initialisierung und Pflege behandelt sowie die Gefahren, die es zu meiden gilt. Die Kernaussagen sollten Sie auch auf Ihre eigenen Designs anwenden: Fragen Sie sich immer als Erstes, woher Ihre »Zufallsbits« kommen.

In Kapitel 4 (»AES«) geht es um das Design der Blockchiffre AES, die Abwägungen zur Implementierung, Gefahren durch Nebenkanäle und die Verwendungsmodi. In diesem Kapitel werfen wir jedoch nur einen flüchtigen Blick auf das Design und konzentrieren uns mehr auf wichtige Designelemente für Implementierer und wie Sie sie in verschiedenen Konfigurationen nutzen können. Als eine der Designgefahren wird hier der Datencache-Nebenkanalangriff von Bernstein besprochen. Den Abschluss des Kapitels bildet eine Erörterung der Verwendungsmodi CBC und CTR, wobei wir uns insbesondere darauf konzentrieren, für welche Probleme sie geeignet sind, wie sie initialisiert werden und welche Gefahren sich bei ihrer Verwendung ergeben.

Kapitel 5 (»Hashfunktionen«) beschreibt die Einweg-Hashfunktionen aus den Familien SHA-1 und SHA-2 des NIST. Dabei sehen wir uns erst die Designs und dann die Abwägungen zur Implementierung an. Zum Thema dieses Kapitels gehören auch die Kollisionsresistenz, Beispiele für Exploits und bekannte Muster falscher Verwendung.

Kapitel 6 (»MAC-Algorithmen«) ist den MAC-Algorithmen (Message Authentication Code) HMAC und CMAC gewidmet, die aus Hash- bzw. Chiffrefunktionen aufgebaut sind. Jeder der Modi wird nacheinander vorgestellt, wobei wir das Design, die Implementierung, die Ziele und die Gefahren der Nutzung besprechen. Besonderes Augenmerk liegt auf der Verhinderung von Replay-Angriffen mithilfe von Countern und Timern, um sowohl Online- als auch Offlineszenarien abzudecken.

Kapitel 7 (»Verschlüsselungs- und Authentifizierungsmodi«) behandelt die Verschlüsselungs- und Authentifizierungsmodi GCM und CCM der IEEE bzw. des NIST. Mit beiden wurden neue Prinzipien in kryptografische Funktionen eingeführt, insbesondere »zusätzliche Authentifizierungsdaten«, also Nachrichtendaten, die authentifiziert, aber nicht verschlüsselt werden, was der Verwendung kryptografischer Algorithmen eine ganz neue Dimension verleiht. Anschließend wird das Design von GCM und CCM ausführlich besprochen. Dank seiner rohen mathematischen Elemente gibt es insbesondere für GCM effiziente tabellengestützte Implementierungen, die wir uns hier ansehen. Ebenso wie im MAC-Kapitel liegt der Schwerpunkt auf Replay-Angriffen. Auch die Initialisierungstechniken werden ausführlich behandelt. Die Modi GCM und LRW sind in gewisser Hinsicht verwandt, da sie beide eine bestimmte Multiplikation verwenden. Daher sollten Sie auf jeden Fall die Erörterung von LRW in Kapitel 4 gelesen haben, bevor Sie sich an dieses Kapitel machen.

Kapitel 8 (»Langzahlarithmetik«) bespricht die Techniken zur Bearbeitung großer Integer, wie sie etwa in Algorithmen für öffentliche Schlüssel verwendet werden. Dabei geht es hauptsächlich um die Flaschenhals-Algorithmen, denen sich Entwickler bei ihren Routineaufgaben gegenübersehen. Um mehr über das Thema zu erfahren, sollten Sie den ergänzenden Text »Multi-Precision Math« auf der Website lesen. Der Schwerpunkt dieses Kapitels liegt auf schneller Multiplikation, Quadrierung, Reduktion und Potenzierung sowie der verschiedenen Abwägungen in der Praxis. Vergleiche der Codegröße und der Leistung sollen Ihnen als Richtschnur dienen, um das Design an die Laufzeiteinschränkungen anzupassen. Dieses Kapitel streift auch einige Gebiete der Zahlentheorie, die für die Lektüre des neunten Kapitels erforderlich sind.

Kapitel 9 (»Algorithmen für öffentliche Schlüssel«) gibt eine Einführung in die Kryptografie mit öffentlichen Schlüsseln. Als Erstes werden dabei der Algorithmus RSA und die zugehörigen Auffüllverfahren von PKCS1 vorgestellt. Sie lernen die verschiedenen Timingangriffe und die entsprechenden Gegenmaßnahmen kennen. Anschließend werden die ECC-Algorithmen EC-DH und EC-DSA für öffentliche Schlüssel besprochen, die beide die elliptischen Kurven des NIST nutzen. Dabei sehen wir uns auch die ANSI-Standards X9.62 und X9.63 an. In diesem Kapitel werden auch einige neue mathematische Grundlagen eingeführt, nämlich verschiedene Multiplikatoren für die Punkte elliptischer Kurven, wobei wir auch auf die jeweiligen Abwägungen zur Leistung eingehen. Um tiefere Kenntnisse über die Mathematik elliptischer Kurven zu gewinnen, sollten Sie das Buch »Guide to Elliptic Curve Cryptography« lesen.

1.9 Häufig gestellte Fragen

Die folgenden »häufig gestellten Fragen« mit den Antworten von den Autoren dieses Buchs sollen Ihnen dabei helfen, Ihr Verständnis des in diesem Kapitel vorgestellten Stoffs zu prüfen, und Sie bei der Umsetzung dieser Prinzipien in der Praxis unterstützen.

F: Wann sollte die Entwicklung des Bedrohungsmodells beginnen?

A: Das Bedrohungsmodell sollte zusammen mit dem Projekt entwickelt werden. Allerdings kann es sein, dass Aspekte wie Laufzeiteinschränkungen nicht im Voraus bekannt sind, was die Genauigkeit des Modells einschränkt.

F: Hat es irgendwelche Vorteile, wenn man seine eigenen kryptografischen Algorithmen entwickelt, anstatt Algorithmen wie AES oder SHA-2 zu verwenden?

A: Normalerweise nicht, was die Sicherheit angeht. Es ist zwar durchaus möglich, dass ein eigener Algorithmus für eine bestimmte Plattform effizienter ist, doch sollten Sie so etwas vermeiden, da das Produkt dann den Bereich der sicheren Standards verlässt und in eine zweifelhafte Grauzone übergeht. Durch die Verwendung der bewährten Vorgehensweisen, zu denen auch die Nutzung von Standardalgorithmen gehört, schränken Sie auch Ihre Haftung ein.

F: Worum handelt es sich bei zertifizierter Kryptografie oder der FIPS-Zertifizierung?

A: Bei der FIPS-Zertifizierung (*http://csrc.nist.gov/cryptval/*) wird eine binäre oder physische Implementierung eines Algorithmus von FIPS-lizenzierten Validierungsstellen überprüft. Beim Bestehen dieser Prüfung wird ein Zertifikat ausgestellt. Die Prüfung gilt für eine bestimmte Implementierung. Es ist nicht möglich, ein Design oder Quellcode zu zertifizieren. Verschiedene Zertifizierungsgrade stehen bereit, je nachdem, wie widerstandsfähig das Produkt sein soll. Bei Ebene 1 wird nur ein Fragenkatalog abgearbeitet, bei Ebene 4 eine physische Untersuchung auf Seitenkanäle durchgeführt.

F: Wo kann ich die Projekte LibTomCrypt und TomsFastMath finden? Auf welchen Plattformen laufen sie? Wie sind sie lizenziert?

A: Die Projekte sind auf *www.libtom.net* untergebracht. Zur Kompilierung sind MSVC und GNU CC geeignet, und unterstützt werden sie auf allen 32- und 64-Bit-Plattformen. Beide Projekte sind gemeinfrei und für alle Zwecke kostenlos.

Stichwortverzeichnis

A

B

C

H

I

J

K

L

M

N

O

P

T

U

Tom St Denis / Simon Johnson

Kryptografie
für Entwickler

Dieses Buch ist das Standardwerk für alle Softwareentwickler, die sich eingehender mit dem Thema Kryptografie auseinandersetzen wollen. Es bietet einen umfassenden Einblick in die Themengebiete Nachrichtenauthentifizierungscodes, Verschlüsselungstechniken, Public-Key-Algorithmen und viele mehr. Zudem zeigt es zahlreiche Anwendungsbeispiele dieser kryptografischen Sicherheitsmaßnahmen. Ein Standardwerk für alle, denen die Sicherheit ihrer Software-Entwicklungen am Herzen liegt.

Das Buch beginnt mit einer fundierten Einführung in das Themengebiet Kryptografie. Im zweiten Kapitel wird vermittelt, wie Langzahl-Arithmetik für RSA- und ECC-Public-Key-Algorithmen implementiert werden muss. Weitere Kapitel erläutern die Einsatzgebiete der systemischen Chiffrierung von Einweg-Hash-Funktionen über Nachrichtenauthenitifizierungscodes hin zu kombinierter Authentifizierung und Verschlüsselung.

Jedes Kapitel enthält übersichtliche und anwenderfreundliche Informationen zu den Auswirkungen der Sicherheitsmaßnahmen auf Datengröße, Systemanforderungen und Performance der Software und stellt deutlich heraus, welche kryptografischen Herausforderungen mit den spezifischen Themen angegangen beziehungsweise gelöst werden.

Aus dem Inhalt:

Über den Autor:

Tom St Denis ist Softwareentwickler und mit seinen kryptografischen Bibliotheken (LibTom.net) bekannt geworden. In den vergangenen Jahren hat er sich der Entwicklung von Open-Source-Kryptografie verschrieben und deren Verbreitung maßgeblich vorangetrieben. Bei der Elliptic Semiconductor Inc. entwickelte er Softwarebibliotheken für eingebettete Systeme und arbeitete eng mit einem Team von Hardware-Ingenieuren zusammen, um eine Best-of-Breed-Hardware- und Software-Kombination zu erschaffen.

Besuchen Sie
unsere Website
www.franzis.de

FRANZIS