

Keyword

☒ Brief ☐ Full

[Advanced Search](#)

[Search Tips](#)

To access the contents, click the chapter and section titles.

Applied Cryptography, Second Edition: Protocols, Algorithms, and Source Code in C (cloth)

(Publisher: John Wiley & Sons, Inc.)

Author(s): Bruce Schneier

ISBN: 0471128457

Publication Date: 01/01/96

Bookmark It

Search this book:

[Foreword by Whitfield Diffie](#)

[Preface](#)

[About the Author](#)

[Chapter 1—Foundations](#)

[1.1 Terminology](#)

[1.2 Steganography](#)

[1.3 Substitution Ciphers and Transposition Ciphers](#)

[1.4 Simple XOR](#)

[1.5 One-Time Pads](#)

[1.6 Computer Algorithms](#)

[1.7 Large Numbers](#)

[Part I—Cryptographic Protocols](#)

[Chapter 2—Protocol Building Blocks](#)

[2.1 Introduction to Protocols](#)

[2.2 Communications Using Symmetric Cryptography](#)

[2.3 One-Way Functions](#)

[2.4 One-Way Hash Functions](#)

[2.5 Communications Using Public-Key Cryptography](#)

[2.6 Digital Signatures](#)

[2.7 Digital Signatures with Encryption](#)

[2.8 Random and Pseudo-Random-Sequence Generation](#)

Chapter 3—Basic Protocols

[3.1 Key Exchange](#)

[3.2 Authentication](#)

[3.3 Authentication and Key Exchange](#)

[3.4 Formal Analysis of Authentication and Key-Exchange Protocols](#)

[3.5 Multiple-Key Public-Key Cryptography](#)

[3.6 Secret Splitting](#)

[3.7 Secret Sharing](#)

[3.8 Cryptographic Protection of Databases](#)

Chapter 4—Intermediate Protocols

[4.1 Timestamping Services](#)

[4.2 Subliminal Channel](#)

[4.3 Undeniable Digital Signatures](#)

[4.4 Designated Confirmer Signatures](#)

[4.5 Proxy Signatures](#)

[4.6 Group Signatures](#)

[4.7 Fail-Stop Digital Signatures](#)

[4.8 Computing with Encrypted Data](#)

[4.9 Bit Commitment](#)

[4.10 Fair Coin Flips](#)

[4.11 Mental Poker](#)

[4.12 One-Way Accumulators](#)

[4.13 All-or-Nothing Disclosure of Secrets](#)

[4.14 Key Escrow](#)

Chapter 5—Advanced Protocols

[5.1 Zero-Knowledge Proofs](#)

[5.2 Zero-Knowledge Proofs of Identity](#)

[5.3 Blind Signatures](#)

[5.4 Identity-Based Public-Key Cryptography](#)

[5.5 Oblivious Transfer](#)

[5.6 Oblivious Signatures](#)

[5.7 Simultaneous Contract Signing](#)

[5.8 Digital Certified Mail](#)

[5.9 Simultaneous Exchange of Secrets](#)

Chapter 6—Esoteric Protocols

[6.1 Secure Elections](#)

[6.2 Secure Multiparty Computation](#)

[6.3 Anonymous Message Broadcast](#)

[6.4 Digital Cash](#)

Part II—Cryptographic Techniques

Chapter 7—Key Length

[7.1 Symmetric Key Length](#)

[7.2 Public-Key Key Length](#)

[7.3 Comparing Symmetric and Public-Key Key Length](#)

[7.4 Birthday Attacks against One-Way Hash Functions](#)

[7.5 How Long Should a Key Be?](#)

[7.6 Caveat Emptor](#)

Chapter 8—Key Management

[8.1 Generating Keys](#)

[8.2 Nonlinear Keyspaces](#)

[8.3 Transferring Keys](#)

[8.4 Verifying Keys](#)

[8.5 Using Keys](#)

[8.6 Updating Keys](#)

[8.7 Storing Keys](#)

[8.8 Backup Keys](#)

[8.9 Compromised Keys](#)

[8.10 Lifetime of Keys](#)

[8.11 Destroying Keys](#)

[8.12 Public-Key Key Management](#)

Chapter 9—Algorithm Types and Modes

[9.1 Electronic Codebook Mode](#)

[9.2 Block Replay](#)

[9.3 Cipher Block Chaining Mode](#)

[9.4 Stream Ciphers](#)

[9.5 Self-Synchronizing Stream Ciphers](#)

[9.6 Cipher-Feedback Mode](#)

[9.7 Synchronous Stream Ciphers](#)

[9.8 Output-Feedback Mode](#)

[9.9 Counter Mode](#)

[9.10 Other Block-Cipher Modes](#)

[9.11 Choosing a Cipher Mode](#)

[9.12 Interleaving](#)

[9.13 Block Ciphers versus Stream Ciphers](#)

[Chapter 10—Using Algorithms](#)

[10.1 Choosing an Algorithm](#)

[10.2 Public-Key Cryptography versus Symmetric Cryptography](#)

[10.3 Encrypting Communications Channels](#)

[10.4 Encrypting Data for Storage](#)

[10.5 Hardware Encryption versus Software Encryption](#)

[10.6 Compression, Encoding, and Encryption](#)

[10.7 Detecting Encryption](#)

[10.8 Hiding Ciphertext in Ciphertext](#)

[10.9 Destroying Information](#)

[Part III—Cryptographic Algorithms](#)

[Chapter 11—Mathematical Background](#)

[11.1 Information Theory](#)

[11.2 Complexity Theory](#)

[11.3 Number Theory](#)

[11.4 Factoring](#)

[11.5 Prime Number Generation](#)

[11.6 Discrete Logarithms in a Finite Field](#)

[Chapter 12—Data Encryption Standard \(DES\)](#)

[12.1 Background](#)

[12.2 Description of DES](#)

[12.3 Security of DES](#)

[12.4 Differential and Linear Cryptanalysis](#)

[12.5 The Real Design Criteria](#)

[12.6 DES Variants](#)

[12.7 How Secure Is DES Today?](#)

Chapter 13—Other Block Ciphers

[13.1 Lucifer](#)

[13.2 Madryga](#)

[13.3 NewDES](#)

[13.4 FEAL](#)

[13.5 REDOC](#)

[13.6 LOKI](#)

[13.7 Khufu and Khafre](#)

[13.8 RC2](#)

[13.9 IDEA](#)

[13.10 MMB](#)

[13.11 CA-1.1](#)

[13.12 Skipjack](#)

Chapter 14—Still Other Block Ciphers

[14.1 GOST](#)

[14.2 CAST](#)

[14.3 Blowfish](#)

[14.4 SAFER](#)

[14.5 3-Way](#)

[14.6 Crab](#)

[14.7 SXAL8/MBAL](#)

[14.8 RC5](#)

[14.9 Other Block Algorithms](#)

[14.10 Theory of Block Cipher Design](#)

[14.11 Using one-Way Hash Functions](#)

[14.12 Choosing a Block Algorithm](#)

Chapter 15—Combining Block Ciphers

[15.1 Double Encryption](#)

[15.2 Triple Encryption](#)

[15.3 Doubling the Block Length](#)

[15.4 Other Multiple Encryption Schemes](#)

[15.5 CDMF Key Shortening](#)

[15.6 Whitening](#)

15.7 Cascading Multiple Block Algorithms

15.8 Combining Multiple Block Algorithms

Chapter 16—Pseudo-Random-Sequence Generators and Stream Ciphers

16.1 Linear Congruential Generators

16.2 Linear Feedback Shift Registers

16.3 Design and Analysis of Stream Ciphers

16.4 Stream Ciphers Using LFSRs

16.5 A5

16.6 Hughes XPD/KPD

16.7 Nanoteg

16.8 Rambutan

16.9 Additive Generators

16.10 Gifford

16.11 Algorithm M

16.12 PKZIP

Chapter 17—Other Stream Ciphers and Real Random-Sequence Generators

17.1 RC4

17.2 SEAL

17.3 WAKE

17.4 Feedback with Carry Shift Registers

17.5 Stream Ciphers Using FCSRs

17.6 Nonlinear-Feedback Shift Registers

17.7 Other Stream Ciphers

17.8 System-Theoretic Approach to Stream-Cipher Design

17.9 Complexity-Theoretic Approach to Stream-Cipher Design

17.10 Other Approaches to Stream-Cipher Design

17.11 Cascading Multiple Stream Ciphers

17.12 Choosing a Stream Cipher

17.13 Generating Multiple Streams from a Single Pseudo-Random-Sequence Generator

17.14 Real Random-Sequence Generators

Chapter 18—One-Way Hash Functions

18.1 Background

18.2 Snefru

18.3 N- Hash

18.4 MD4

18.5 MD5

18.6 MD2

18.7 Secure Hash Algorithm (SHA)

18.8 RIPE-MD

18.9 HAVAL

18.10 Other One-Way Hash Functions

18.11 One-Way Hash Functions Using Symmetric Block Algorithms

18.12 Using Public-Key Algorithms

18.13 Choosing a One-Way Hash Function

18.14 Message Authentication Codes

Chapter 19—Public-Key Algorithms

19.1 Background

19.2 Knapsack Algorithms

19.3 RSA

19.4 Pohlig-Hellman

19.5 Rabin

19.6 ElGamal

19.7 McEliece

19.8 Elliptic Curve Cryptosystems

19.9 LUC

19.10 Finite Automaton Public-Key Cryptosystems

Chapter 20—Public-Key Digital Signature Algorithms

20.1 Digital Signature Algorithm (DSA)

20.2 DSA Variants

20.3 Gost Digital Signature Algorithm

20.4 Discrete Logarithm Signature Schemes

20.5 Ong-Schnorr-Shamir

20.6 ESIGN

20.7 Cellular Automata

20.8 Other Public-Key Algorithms

Chapter 21—Identification Schemes

21.1 Feige-Fiat-Shamir

21.2 Guillou-Quisquater

21.3 Schnorr

21.4 Converting Identification Schemes to Signature Schemes

Chapter 22—Key-Exchange Algorithms

22.1 Diffie-Hellman

22.2 Station-to-Station Protocol

22.3 Shamir's Three-Pass Protocol

22.4 COMSET

22.5 Encrypted Key Exchange

22.6 Fortified Key Negotiation

22.7 Conference Key Distribution and Secret Broadcasting

Chapter 23—Special Algorithms for Protocols

23.1 Multiple-Key Public-Key Cryptography

23.2 Secret-Sharing Algorithms

23.3 Subliminal Channel

23.4 Undeniable Digital Signatures

23.5 Designated Confirmer Signatures

23.6 Computing with Encrypted Data

23.7 Fair Coin Flips

23.8 One-Way Accumulators

23.9 All-or-Nothing Disclosure of Secrets

23.10 Fair and Failsafe Cryptosystems

23.11 Zero-Knowledge Proofs of Knowledge

23.12 Blind Signatures

23.13 Oblivious Transfer

23.14 Secure Multiparty Computation

23.15 Probabilistic Encryption

23.16 Quantum Cryptography

Part IV—The Real World

Chapter 24—Example Implementations

24.1 IBM Secret-Key Management Protocol

24.2 MITRENET

24.3 ISDN

24.4 STU-III

24.5 Kerberos

24.6 KryptoKnight

24.7 SESAME

24.8 IBM Common Cryptographic Architecture

24.9 ISO Authentication Framework

24.10 Privacy-Enhanced Mail (PEM)

24.11 Message Security Protocol (MSP)

24.12 Pretty Good Privacy (PGP)

24.13 Smart Cards

24.14 Public-Key Cryptography Standards (PKCS)

24.15 Universal Electronic Payment System (UEPS)

24.16 Clipper

24.17 Capstone

24.18 AT&T Model 3600 Telephone Security Device (TSD)

Chapter 25—Politics

25.1 National Security Agency (NSA)

25.2 National Computer Security Center (NCSC)

25.3 National Institute of Standards and Technology (NIST)

25.4 RSA Data Security, Inc.

25.5 Public Key Partners

25.6 International Association for Cryptologic Research (IACR)

25.7 RACE Integrity Primitives Evaluation (RIPE)

25.8 Conditional Access for Europe (CAFE)

25.9 ISO/IEC 9979

25.10 Professional, Civil Liberties, and Industry Groups

25.11 Sci.crypt

25.12 Cypherpunks

[25.13 Patents](#)

[25.14 U.S. Export Rules](#)

[25.15 Foreign Import and Export of Cryptography](#)

[25.16 Legal Issues](#)

[Afterword by Matt Blaze](#)

[Part V—Source Code](#)

[References](#)

[Index](#)

[Products](#) | [Contact Us](#) | [About Us](#) | [Privacy](#) | [Ad Info](#) | [Home](#)

Use of this site is subject to certain [Terms & Conditions](#), [Copyright © 1996-2000 EarthWeb Inc.](#)

All rights reserved. Reproduction whole or in part in any form or medium without express written [permission](#) of EarthWeb is prohibited. Read EarthWeb's [privacy](#) statement.

 SEARCH
ITKNOWLEDGE

☒ Brief
 ☐ Full

[Advanced Search](#)

[Search Tips](#)

 BROWSE
BY TOPIC

To access the contents, click the chapter and section titles.

Applied Cryptography, Second Edition: Protocols, Algorithms, and Source Code in C (cloth)

(Publisher: John Wiley & Sons, Inc.)

Author(s): Bruce Schneier

ISBN: 0471128457

Publication Date: 01/01/96

Bookmark It

Search this book:

[Previous](#)
[Table of Contents](#)
[Next](#)

Foreword By Whitfield Diffie

The literature of cryptography has a curious history. Secrecy, of course, has always played a central role, but until the First World War, important developments appeared in print in a more or less timely fashion and the field moved forward in much the same way as other specialized disciplines. As late as 1918, one of the most influential cryptanalytic papers of the twentieth century, William F. Friedman's monograph *The Index of Coincidence and Its Applications in Cryptography*, appeared as a research report of the private Riverbank Laboratories [577]. And this, despite the fact that the work had been done as part of the war effort. In the same year Edward H. Hebern of Oakland, California filed the first patent for a rotor machine [710], the device destined to be a mainstay of military cryptography for nearly 50 years.

After the First World War, however, things began to change. U.S. Army and Navy organizations, working entirely in secret, began to make fundamental advances in cryptography. During the thirties and forties a few basic papers did appear in the open literature and several treatises on the subject were published, but the latter were farther and farther behind the state of the art. By the end of the war the transition was complete. With one notable exception, the public literature had died. That exception was Claude Shannon's paper "The Communication Theory of Secrecy Systems," which appeared in the *Bell System Technical Journal* in 1949 [1432]. It was similar to Friedman's 1918 paper, in that it grew out of wartime work of Shannon's. After the Second World War ended it was declassified, possibly by mistake.

From 1949 until 1967 the cryptographic literature was barren. In that year a different sort of contribution appeared: David Kahn's history, *The Codebreakers* [794]. It didn't contain any new technical ideas, but it did contain a remarkably complete history of what had gone before, including

mention of some things that the government still considered secret. The significance of *The Codebreakers* lay not just in its remarkable scope, but also in the fact that it enjoyed good sales and made tens of thousands of people, who had never given the matter a moment's thought, aware of cryptography. A trickle of new cryptographic papers began to be written.

At about the same time, Horst Feistel, who had earlier worked on identification friend or foe devices for the Air Force, took his lifelong passion for cryptography to the IBM Watson Laboratory in Yorktown Heights, New York. There, he began development of what was to become the U.S. Data Encryption Standard; by the early 1970s several technical reports on this subject by Feistel and his colleagues had been made public by IBM [1482,1484,552].

This was the situation when I entered the field in late 1972. The cryptographic literature wasn't abundant, but what there was included some very shiny nuggets.

Cryptology presents a difficulty not found in normal academic disciplines: the need for the proper interaction of cryptography and cryptanalysis. This arises out of the fact that in the absence of real communications requirements, it is easy to propose a system that appears unbreakable. Many academic designs are so complex that the would-be cryptanalyst doesn't know where to start; exposing flaws in these designs is far harder than designing them in the first place. The result is that the competitive process, which is one strong motivation in academic research, cannot take hold.

When Martin Hellman and I proposed public-key cryptography in 1975 [496], one of the indirect aspects of our contribution was to introduce a problem that does not even appear easy to solve. Now an aspiring cryptosystem designer could produce something that would be recognized as clever—something that did more than just turn meaningful text into nonsense. The result has been a spectacular increase in the number of people working in cryptography, the number of meetings held, and the number of books and papers published.

In my acceptance speech for the Donald E. Fink award—given for the best expository paper to appear in an IEEE journal—which I received jointly with Hellman in 1980, I told the audience that in writing “Privacy and Authentication,” I had an experience that I suspected was rare even among the prominent scholars who populate the IEEE awards ceremony: I had written the paper I had wanted to study, but could not find, when I first became seriously interested in cryptography. Had I been able to go to the Stanford bookstore and pick up a modern cryptography text, I would probably have learned about the field years earlier. But the only things available in the fall of 1972 were a few classic papers and some obscure technical reports.

The contemporary researcher has no such problem. The problem now is choosing where to start among the thousands of papers and dozens of books. The contemporary researcher, yes, but what about the contemporary programmer or engineer who merely wants to use cryptography? Where does that person turn? Until now, it has been necessary to spend long hours hunting out and then studying the research literature before being able to design the sort of cryptographic utilities glibly described in popular articles.

This is the gap that Bruce Schneier's *Applied Cryptography* has come to fill. Beginning with the objectives of communication security and elementary examples of programs used to achieve these objectives, Schneier gives us a panoramic view of the fruits of 20 years of public research. The title says it all; from the mundane objective of having a secure conversation the very first time you call someone to the possibilities of digital money and cryptographically secure elections, this is where you'll find it.

Not satisfied that the book was about the real world merely because it went all the way down to the code, Schneier has included an account of the world in which cryptography is developed and applied, and discusses entities ranging from the International Association for Cryptologic Research to the NSA.

When public interest in cryptography was just emerging in the late seventies and early eighties, the National Security Agency (NSA), America's official cryptographic organ, made several attempts to quash it. The first was a letter from a long-time NSA employee allegedly, avowedly, and apparently acting on his own. The letter was sent to the IEEE and warned that the publication of cryptographic material was a violation of the International Traffic in Arms Regulations (ITAR). This viewpoint turned out not even to be supported by the regulations themselves—which contained an explicit exemption for published material—but gave both the public practice of cryptography and the 1977 Information Theory Workshop lots of unexpected publicity.

A more serious attempt occurred in 1980, when the NSA funded the American Council on Education to examine the issue with a view to persuading Congress to give it legal control of publications in the field of cryptography. The results fell far short of NSA's ambitions and resulted in a program of voluntary review of cryptographic papers; researchers were requested to ask the NSA's opinion on whether disclosure of results would adversely affect the national interest before publication.

As the eighties progressed, pressure focused more on the practice than the study of cryptography. Existing laws gave the NSA the power, through the Department of State, to regulate the export of cryptographic equipment. As business became more and more international and the American fraction of the world market declined, the pressure to have a single product in both domestic and offshore markets increased. Such single products were subject to export control and thus the NSA acquired substantial influence not only over what was exported, but also over what was sold in the United States.

As this is written, a new challenge confronts the public practice of cryptography. The government has augmented the widely published and available Data Encryption Standard, with a secret algorithm implemented in tamper-resistant chips. These chips will incorporate a codified mechanism of government monitoring. The negative aspects of this "key-escrow" program range from a potentially disastrous impact on personal privacy to the high cost of having to add hardware to products that had previously encrypted in software. So far key escrow products are enjoying less than stellar sales and the scheme has attracted widespread negative comment, especially from the independent cryptographers. Some people, however, see more future in programming than politicking and have redoubled their efforts to provide the

world with strong cryptography that is accessible to public scrutiny.

A sharp step back from the notion that export control law could supersede the First Amendment seemed to have been taken in 1980 when the *Federal Register* announcement of a revision to ITAR included the statement: “...provision has been added to make it clear that the regulation of the export of technical data does not purport to interfere with the First Amendment rights of individuals.” But the fact that tension between the First Amendment and the export control laws has not gone away should be evident from statements at a conference held by RSA Data Security. NSA’s representative from the export control office expressed the opinion that people who published cryptographic programs were “in a grey area” with respect to the law. If that is so, it is a grey area on which the first edition of this book has shed some light. Export applications for the book itself have been granted, with acknowledgement that published material lay beyond the authority of the Munitions Control Board. Applications to export the enclosed programs on disk, however, have been denied.

The shift in the NSA’s strategy, from attempting to control cryptographic research to tightening its grip on the development and deployment of cryptographic products, is presumably due to its realization that all the great cryptographic papers in the world do not protect a single bit of traffic. Sitting on the shelf, this volume may be able to do no better than the books and papers that preceded it, but sitting next to a workstation, where a programmer is writing cryptographic code, it just may.

Whitfield Diffie
Mountain View,
CA

Previous	Table of Contents	Next
--------------------------	-----------------------------------	----------------------

[Products](#) | [Contact Us](#) | [About Us](#) | [Privacy](#) | [Ad Info](#) | [Home](#)

Use of this site is subject to certain [Terms & Conditions](#), [Copyright © 1996-2000 EarthWeb Inc.](#)

All rights reserved. Reproduction whole or in part in any form or medium without express written [permission](#) of EarthWeb is prohibited. Read EarthWeb's [privacy](#) statement.

SEARCH
ITKNOWLEDGE

Go!

Keyword

☒ Brief ☐ Full

[Advanced Search](#)

[Search Tips](#)

BROWSE
BY TOPIC

To access the contents, click the chapter and section titles.

Applied Cryptography, Second Edition: Protocols, Algorithms, and Source Code in C (cloth)

(Publisher: John Wiley & Sons, Inc.)

Author(s): Bruce Schneier

ISBN: 0471128457

Publication Date: 01/01/96

[Bookmark It](#)

Search this book:

Go!

[Previous](#) [Table of Contents](#) [Next](#)

Preface

There are two kinds of cryptography in this world: cryptography that will stop your kid sister from reading your files, and cryptography that will stop major governments from reading your files. This book is about the latter.

If I take a letter, lock it in a safe, hide the safe somewhere in New York, then tell you to read the letter, that's not security. That's obscurity. On the other hand, if I take a letter and lock it in a safe, and then give you the safe along with the design specifications of the safe and a hundred identical safes with their combinations so that you and the world's best safecrackers can study the locking mechanism—and you still can't open the safe and read the letter—that's security.

For many years, this sort of cryptography was the exclusive domain of the military. The United States' National Security Agency (NSA), and its counterparts in the former Soviet Union, England, France, Israel, and elsewhere, have spent billions of dollars in the very serious game of securing their own communications while trying to break everyone else's. Private individuals, with far less expertise and budget, have been powerless to protect their own privacy against these governments.

During the last 20 years, public academic research in cryptography has exploded. While classical cryptography has been long used by ordinary citizens, computer cryptography was the exclusive domain of the world's militaries since World War II. Today, state-of-the-art computer cryptography is practiced outside the secured walls of the military agencies. The layperson can now employ security practices that can protect against the most powerful of adversaries—security that may protect against military agencies for years to come.

Do average people really need this kind of security? Yes. They may be planning a political campaign, discussing taxes, or having an illicit affair. They may be designing a new product, discussing a marketing strategy, or planning a hostile business takeover. Or they may be living in a country that does not respect the rights of privacy of its citizens. They may be doing something that they feel shouldn't be illegal, but is. For whatever reason, the data and communications are personal, private, and no one else's business.

This book is being published in a tumultuous time. In 1994, the Clinton administration approved the Escrowed Encryption Standard (including the Clipper chip and Fortezza card) and signed the Digital Telephony bill into law. Both of these initiatives try to ensure the government's ability to conduct electronic surveillance.

Some dangerously Orwellian assumptions are at work here: that the government has the right to listen to private communications, and that there is something wrong with a private citizen trying to keep a secret from the government. Law enforcement has always been able to conduct court-authorized surveillance if possible, but this is the first time that the people have been forced to take active measures to *make themselves available* for surveillance. These initiatives are not simply government proposals in some obscure area; they are preemptive and unilateral attempts to usurp powers that previously belonged to the people.

Clipper and Digital Telephony do not protect privacy; they force individuals to unconditionally trust that the government will respect their privacy. The same law enforcement authorities who illegally tapped Martin Luther King Jr.'s phones can easily tap a phone protected with Clipper. In the recent past, local police authorities have either been charged criminally or sued civilly in numerous jurisdictions—Maryland, Connecticut, Vermont, Georgia, Missouri, and Nevada—for conducting illegal wiretaps. It's a poor idea to deploy a technology that could some day facilitate a police state.

The lesson here is that it is insufficient to protect ourselves with laws; we need to protect ourselves with mathematics. Encryption is too important to be left solely to governments.

This book gives you the tools you need to protect your own privacy; cryptography products may be declared illegal, but the information will never be.

How to Read This Book

I wrote *Applied Cryptography* to be both a lively introduction to the field of cryptography and a comprehensive reference. I have tried to keep the text readable without sacrificing accuracy. This book is not intended to be a mathematical text. Although I have not deliberately given any false information, I do play fast and loose with theory. For those interested in formalism, there are copious references to the academic literature.

Chapter 1 introduces cryptography, defines many terms, and briefly discusses precomputer cryptography.

Chapters 2 through 6 (Part I) describe cryptographic protocols: what people can do with cryptography. The protocols range from the simple (sending encrypted messages from one person to another) to the complex (flipping a coin over the telephone) to the esoteric (secure and anonymous digital money exchange). Some of these protocols are obvious; others are almost amazing. Cryptography can solve a lot of problems that most people never realized it could.

Chapters 7 through 10 (Part II) discuss cryptographic techniques. All four chapters in this section are important for even the most basic uses of cryptography. Chapters 7 and 8 are about keys: how long a key should be in order to be secure, how to generate keys, how to store keys, how to dispose of keys, and so on. Key management is the hardest part of cryptography and often the Achilles' heel of an otherwise secure system. Chapter 9 discusses different ways of using cryptographic algorithms, and Chapter 10 gives the odds and ends of algorithms: how to choose, implement, and use algorithms.

Chapters 11 through 23 (Part III) list algorithms. Chapter 11 provides the mathematical background. This chapter is only required if you are interested in public-key algorithms. If you just want to implement DES (or something similar), you can skip ahead. Chapter 12 discusses DES: the algorithm, its history, its security, and some variants. Chapters 13, 14, and 15 discuss other block algorithms; if you want something more secure than DES, skip to the section on IDEA and triple-DES. If you want to read about a bunch of algorithms, some of which may be more secure than DES, read the whole chapter. Chapters 16 and 17 discuss stream algorithms. Chapter 18 focuses on one-way hash functions; MD5 and SHA are the most common, although I discuss many more. Chapter 19 discusses public-key encryption algorithms, Chapter 20 discusses public-key digital signature algorithms, Chapter 21 discusses public-key identification algorithms, and Chapter 22 discusses public-key key exchange algorithms. The important algorithms are RSA, DSA, Fiat-Shamir, and Diffie-Hellman, respectively. Chapter 23 has more esoteric public-key algorithms and protocols; the math in this chapter is quite complicated, so wear your seat belt.

Chapters 24 and 25 (Part IV) turn to the real world of cryptography. Chapter 24 discusses some of the current implementations of these algorithms and protocols, while Chapter 25 touches on some of the political issues surrounding cryptography. These chapters are by no means intended to be comprehensive.

Also included are source code listings for 10 algorithms discussed in Part III. I was unable to include all the code I wanted to due to space limitations, and cryptographic source code cannot otherwise be exported. (Amazingly enough, the State Department allowed export of the first edition of this book with source code, but denied export for a computer disk with the exact same source code on it. Go figure.) An associated source code disk set includes much more source code than I could fit in this book; it is probably the largest collection of cryptographic source code outside a military institution. I can only send source code disks to U.S. and Canadian citizens living in the U.S. and Canada, but hopefully that will change someday. If you are interested in implementing or playing with the cryptographic algorithms in this book, get the disk. See the

last page of the book for details.

One criticism of this book is that its encyclopedic nature takes away from its readability. This is true, but I wanted to provide a single reference for those who might come across an algorithm in the academic literature or in a product. For those who are more interested in a tutorial, I apologize. A lot is being done in the field; this is the first time so much of it has been gathered between two covers. Even so, space considerations forced me to leave many things out. I covered topics that I felt were important, practical, or interesting. If I couldn't cover a topic in depth, I gave references to articles and papers that did.

I have done my best to hunt down and eradicate all errors in this book, but many have assured me that it is an impossible task. Certainly, the second edition has far fewer errors than the first. An errata listing is available from me and will be periodically posted to the Usenet newsgroup sci.crypt. If any reader finds an error, please let me know. I'll send the first person to find each error in the book a free copy of the source code disk.

Previous	Table of Contents	Next
--------------------------	-----------------------------------	----------------------

[Products](#) | [Contact Us](#) | [About Us](#) | [Privacy](#) | [Ad Info](#) | [Home](#)

Use of this site is subject to certain [Terms & Conditions](#), [Copyright © 1996-2000 EarthWeb Inc.](#)

All rights reserved. Reproduction whole or in part in any form or medium without express written [permission](#) of EarthWeb is prohibited. Read EarthWeb's [privacy](#) statement.

 SEARCH
ITKNOWLEDGE

☒ Brief
 ☐ Full

[Advanced Search](#)

[Search Tips](#)

 BROWSE
BY TOPIC

To access the contents, click the chapter and section titles.

Applied Cryptography, Second Edition: Protocols, Algorithms, and Source Code in C (cloth)

(Publisher: John Wiley & Sons, Inc.)

Author(s): Bruce Schneier

ISBN: 0471128457

Publication Date: 01/01/96

Bookmark It

Search this book:

[Previous](#)
[Table of Contents](#)
[Next](#)

About the Author

BRUCE SCHNEIER is president of Counterpane Systems, an Oak Park, Illinois consulting firm specializing in cryptography and computer security. Bruce is also the author of *E-Mail Security* (John Wiley & Sons, 1995) and *Protect Your Macintosh* (Peachpit Press, 1994); and has written dozens of articles on cryptography for major magazines. He is a contributing editor to *Dr. Dobb's Journal*, where he edits the "Algorithms Alley" column, and a contributing editor to *Computer and Communications Security Reviews*. Bruce serves on the board of directors of the International Association for Cryptologic Research, is a member of the Advisory Board for the Electronic Privacy Information Center, and is on the program committee for the New Security Paradigms Workshop. In addition, he finds time to give frequent lectures on cryptography, computer security, and privacy.

Acknowledgments

The list of people who had a hand in this book may seem unending, but all are worthy of mention. I would like to thank Don Alvarez, Ross Anderson, Dave Balenson, Karl Barrus, Steve Bellovin, Dan Bernstein, Eli Biham, Joan Boyar, Karen Cooper, Whit Diffie, Joan Feigenbaum, Phil Karn, Neal Koblitz, Xuejia Lai, Tom Leran, Mike Markowitz, Ralph Merkle, Bill Patton, Peter Pearson, Charles Pfleeger, Ken Pizzini, Bart Preneel, Mark Riordan, Joachim Schurman, and Marc Schwartz for reading and editing all or parts of the first edition; Marc Vaclair for translating the first edition into French; Abe Abraham, Ross Anderson, Dave Banisar, Steve Bellovin, Eli Biham, Matt Bishop, Matt Blaze, Gary Carter, Jan Camenisch, Claude CrŽpeau, Joan Daemen, Jorge Davila, Ed Dawson, Whit Diffie, Carl Ellison, Joan Feigenbaum, Niels Ferguson, Matt Franklin, Rosario Gennaro, Dieter

Gollmann, Mark Goresky, Richard Graveman, Stuart Haber, Jingman He, Bob Hogue, Kenneth Iversen, Markus Jakobsson, Burt Kaliski, Phil Karn, John Kelsey, John Kennedy, Lars Knudsen, Paul Kocher, John Ladwig, Xuejia Lai, Arjen Lenstra, Paul Leyland, Mike Markowitz, Jim Massey, Bruce McNair, William Hugh Murray, Roger Needham, Clif Neuman, Kaisa Nyberg, Luke O'Connor, Peter Pearson, RenŽ Peralta, Bart Preneel, Yisrael Radai, Matt Robshaw, Michael Roe, Phil Rogaway, Avi Rubin, Paul Rubin, Selwyn Russell, Kazue Sako, Mahmoud Salmasizadeh, Markus Stadler, Dmitry Titov, Jimmy Upton, Marc Vauclair, Serge Vaudenay, Gideon Yuval, Glen Zorn, and several anonymous government employees for reading and editing all or parts of the second edition; Lawrie Brown, Leisa Condie, Joan Daemen, Peter Gutmann, Alan Insley, Chris Johnston, John Kelsey, Xuejia Lai, Bill Leininger, Mike Markowitz, Richard Outerbridge, Peter Pearson, Ken Pizzini, Colin Plumb, RSA Data Security, Inc., Michael Roe, Michael Wood, and Phil Zimmermann for providing source code; Paul MacNerland for creating the figures for the first edition; Karen Cooper for copyediting the second edition; Beth Friedman for proofreading the second edition; Carol Kennedy for indexing the second edition; the readers of sci.crypt and the Cypherpunks mailing list for commenting on ideas, answering questions, and finding errors in the first edition; Randy Seuss for providing Internet access; Jeff Duntemann and Jon Erickson for helping me get started; assorted random Insleys for the impetus, encouragement, support, conversations, friendship, and dinners; and AT&T Bell Labs for firing me and making this all possible. All these people helped to create a far better book than I could have created alone.

Bruce Schneier
Oak Park, Ill.
schneier@counterpane.com

Previous	Table of Contents	Next
--------------------------	-----------------------------------	----------------------

[Products](#) | [Contact Us](#) | [About Us](#) | [Privacy](#) | [Ad Info](#) | [Home](#)

Use of this site is subject to certain [Terms & Conditions](#), Copyright © 1996-2000 EarthWeb Inc.

All rights reserved. Reproduction whole or in part in any form or medium without express written [permission](#) of EarthWeb is prohibited. Read EarthWeb's [privacy](#) statement.

 SEARCH
ITKNOWLEDGE

☒ Brief
 ☐ Full

[Advanced Search](#)

[Search Tips](#)

 **BROWSE**
BY TOPIC

To access the contents, click the chapter and section titles.

Applied Cryptography, Second Edition: Protocols, Algorithms, and Source Code in C (cloth)

(Publisher: John Wiley & Sons, Inc.)

Author(s): Bruce Schneier

ISBN: 0471128457

Publication Date: 01/01/96

Bookmark It

Search this book:

[Previous](#)
[Table of Contents](#)
[Next](#)

Chapter 1

Foundations

1.1 Terminology

Sender and Receiver

Suppose a sender wants to send a message to a receiver. Moreover, this sender wants to send the message securely: She wants to make sure an eavesdropper cannot read the message.

Messages and Encryption

A message is **plaintext** (sometimes called cleartext). The process of disguising a message in such a way as to hide its substance is **encryption**. An encrypted message is **ciphertext**. The process of turning ciphertext back into plaintext is **decryption**. This is all shown in Figure 1.1.

(If you want to follow the ISO 7498-2 standard, use the terms “encipher” and “decipher.” It seems that some cultures find the terms “encrypt” and “decrypt” offensive, as they refer to dead bodies.)

The art and science of keeping messages secure is **cryptography**, and it is practiced by **cryptographers**. **Cryptanalysts** are practitioners of **cryptanalysis**, the art and science of breaking ciphertext; that is, seeing through the disguise. The branch of mathematics encompassing both cryptography and cryptanalysis is **cryptology** and its practitioners are **cryptologists**. Modern cryptologists are generally trained in theoretical mathematics—they have to be.



Figure 1.1 *Encryption and Decryption.*

Plaintext is denoted by M , for message, or P , for plaintext. It can be a stream of bits, a text file, a bitmap, a stream of digitized voice, a digital video image...whatever. As far as a computer is concerned, M is simply binary data. (After this chapter, this book concerns itself with binary data and computer cryptography.) The plaintext can be intended for either transmission or storage. In any case, M is the message to be encrypted.

Ciphertext is denoted by C . It is also binary data: sometimes the same size as M , sometimes larger. (By combining encryption with compression, C may be smaller than M . However, encryption does not accomplish this.) The encryption function E , operates on M to produce C . Or, in mathematical notation:

$$E(M) = C$$

In the reverse process, the decryption function D operates on C to produce M :

$$D(C) = M$$

Since the whole point of encrypting and then decrypting a message is to recover the original plaintext, the following identity must hold true:

$$D(E(M)) = M$$

Authentication, Integrity, and Nonrepudiation

In addition to providing confidentiality, cryptography is often asked to do other jobs:

- **Authentication.** It should be possible for the receiver of a message to ascertain its origin; an intruder should not be able to masquerade as someone else.
- **Integrity.** It should be possible for the receiver of a message to verify that it has not been modified in transit; an intruder should not be able to substitute a false message for a legitimate one.
- **Nonrepudiation.** A sender should not be able to falsely deny later that he sent a message.

These are vital requirements for social interaction on computers, and are analogous to face-to-face interactions. That someone is who he says he is...that someone's credentials—whether a driver's license, a medical degree, or a passport—are valid...that a document purporting to come from a person actually came from that person.... These are the things that authentication, integrity, and nonrepudiation provide.

Algorithms and Keys

A **cryptographic algorithm**, also called a **cipher**, is the mathematical function used for encryption and decryption. (Generally, there are two related functions: one for encryption and the other for decryption.)

If the security of an algorithm is based on keeping the way that algorithm

works a secret, it is a **restricted** algorithm. Restricted algorithms have historical interest, but are woefully inadequate by today's standards. A large or changing group of users cannot use them, because every time a user leaves the group everyone else must switch to a different algorithm. If someone accidentally reveals the secret, everyone must change their algorithm.

Even more damning, restricted algorithms allow no quality control or standardization. Every group of users must have their own unique algorithm. Such a group can't use off-the-shelf hardware or software products; an eavesdropper can buy the same product and learn the algorithm. They have to write their own algorithms and implementations. If no one in the group is a good cryptographer, then they won't know if they have a secure algorithm.

Despite these major drawbacks, restricted algorithms are enormously popular for low-security applications. Users either don't realize or don't care about the security problems inherent in their system.

Modern cryptography solves this problem with a **key**, denoted by K . This key might be any one of a large number of values. The range of possible values of the key is called the **keyspace**. Both the encryption and decryption operations use this key (i.e., they are dependent on the key and this fact is denoted by the k subscript), so the functions now become:

$$E_K(M) = C$$

$$D_K(C) = M$$

Those functions have the property that (see Figure 1.2):

$$D_K(E_K(M)) = M$$

Some algorithms use a different encryption key and decryption key (see Figure 1.3). That is, the encryption key, K_1 , is different from the corresponding decryption key, K_2 . In this case:

$$E_{K_1}(M) = C$$

$$D_{K_2}(C) = M$$

$$D_{K_2}(E_{K_1}(M)) = M$$

All of the security in these algorithms is based in the key (or keys); none is based in the details of the algorithm. This means that the algorithm can be published and analyzed. Products using the algorithm can be mass-produced. It doesn't matter if an eavesdropper knows your algorithm; if she doesn't know your particular key, she can't read your messages.

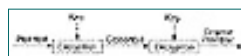


Figure 1.2 Encryption and decryption with a key.



Figure 1.3 Encryption and decryption with two different keys.

A **cryptosystem** is an algorithm, plus all possible plaintexts, ciphertexts, and keys.

Symmetric Algorithms

There are two general types of key-based algorithms: symmetric and public-key. **Symmetric algorithms**, sometimes called conventional algorithms, are algorithms where the encryption key can be calculated from the decryption key and vice versa. In most symmetric algorithms, the encryption key and the decryption key are the same. These algorithms, also called secret-key algorithms, single-key algorithms, or one-key algorithms, require that the sender and receiver agree on a key before they can communicate securely. The security of a symmetric algorithm rests in the key; divulging the key means that anyone could encrypt and decrypt messages. As long as the communication needs to remain secret, the key must remain secret.

Encryption and decryption with a symmetric algorithm are denoted by:

$$E_K(M) = C$$

$$D_K(C) = M$$

Previous	Table of Contents	Next
--------------------------	-----------------------------------	----------------------

[Products](#) | [Contact Us](#) | [About Us](#) | [Privacy](#) | [Ad Info](#) | [Home](#)

Use of this site is subject to certain [Terms & Conditions](#), [Copyright © 1996-2000 EarthWeb Inc.](#)

All rights reserved. Reproduction whole or in part in any form or medium without express written [permission](#) of EarthWeb is prohibited. Read EarthWeb's [privacy](#) statement.

 SEARCH
ITKNOWLEDGE

☒ Brief
 ☐ Full

[Advanced Search](#)

[Search Tips](#)

 BROWSE
BY TOPIC

To access the contents, click the chapter and section titles.

Applied Cryptography, Second Edition: Protocols, Algorithms, and Source Code in C (cloth)

(Publisher: John Wiley & Sons, Inc.)

Author(s): Bruce Schneier

ISBN: 0471128457

Publication Date: 01/01/96

Bookmark It

Search this book:

[Previous](#)
[Table of Contents](#)
[Next](#)

Symmetric algorithms can be divided into two categories. Some operate on the plaintext a single bit (or sometimes byte) at a time; these are called **stream algorithms** or **stream ciphers**. Others operate on the plaintext in groups of bits. The groups of bits are called **blocks**, and the algorithms are called **block algorithms** or **block ciphers**. For modern computer algorithms, a typical block size is 64 bits—large enough to preclude analysis and small enough to be workable. (Before computers, algorithms generally operated on plaintext one character at a time. You can think of this as a stream algorithm operating on a stream of characters.)

Public-Key Algorithms

Public-key algorithms (also called asymmetric algorithms) are designed so that the key used for encryption is different from the key used for decryption. Furthermore, the decryption key cannot (at least in any reasonable amount of time) be calculated from the encryption key. The algorithms are called “public-key” because the encryption key can be made public: A complete stranger can use the encryption key to encrypt a message, but only a specific person with the corresponding decryption key can decrypt the message. In these systems, the encryption key is often called the **public key**, and the decryption key is often called the **private key**. The private key is sometimes also called the secret key, but to avoid confusion with symmetric algorithms, that tag won’t be used here.

Encryption using public key K is denoted by:

$$E_K(M) = C$$

Even though the public key and private key are different, decryption with the corresponding private key is denoted by:

$$D_K(C) = M$$

Sometimes, messages will be encrypted with the private key and decrypted with the public key; this is used in digital signatures (see Section 2.6). Despite the possible confusion, these operations are denoted by, respectively:

$$E_K(M) = C$$

$$D_K(C) = M$$

Cryptanalysis

The whole point of cryptography is to keep the plaintext (or the key, or both) secret from eavesdroppers (also called adversaries, attackers, interceptors, interlopers, intruders, opponents, or simply the enemy). Eavesdroppers are assumed to have complete access to the communications between the sender and receiver.

Cryptanalysis is the science of recovering the plaintext of a message without access to the key. Successful cryptanalysis may recover the plaintext or the key. It also may find weaknesses in a cryptosystem that eventually lead to the previous results. (The loss of a key through noncryptanalytic means is called a **compromise**.)

An attempted cryptanalysis is called an **attack**. A fundamental assumption in cryptanalysis, first enunciated by the Dutchman A. Kerckhoffs in the nineteenth century, is that the secrecy must reside entirely in the key [794]. Kerckhoffs assumes that the cryptanalyst has complete details of the cryptographic algorithm and implementation. (Of course, one would assume that the CIA does not make a habit of telling Mossad about its cryptographic algorithms, but Mossad probably finds out anyway.) While real-world cryptanalysts don't always have such detailed information, it's a good assumption to make. If others can't break an algorithm, even with knowledge of how it works, then they certainly won't be able to break it without that knowledge.

There are four general types of cryptanalytic attacks. Of course, each of them assumes that the cryptanalyst has complete knowledge of the encryption algorithm used:

1. Ciphertext-only attack. The cryptanalyst has the ciphertext of several messages, all of which have been encrypted using the same encryption algorithm. The cryptanalyst's job is to recover the plaintext of as many messages as possible, or better yet to deduce the key (or keys) used to encrypt the messages, in order to decrypt other messages encrypted with the same keys.

$$\text{Given: } C_1 = E_k(P_1), C_2 = E_k(P_2), \dots, C_i = E_k(P_i)$$

$$\text{Deduce: Either } P_1, P_2, \dots, P_i; k; \text{ or an algorithm to infer } P_{i+1} \text{ from } C_{i+1} = E_k(P_{i+1})$$

2. Known-plaintext attack. The cryptanalyst has access not only to the ciphertext of several messages, but also to the plaintext of those messages. His job is to deduce the key (or keys) used to encrypt the messages or an algorithm to decrypt any new messages encrypted with the same key (or keys).

Given: $P_1, C_1 = E_k(P_1), P_2, C_2 = E_k(P_2), \dots, P_i, C_i = E_k(P_i)$

Deduce: Either k , or an algorithm to infer P_{i+1} from $C_{i+1} = E_k(P_{i+1})$

3. Chosen-plaintext attack. The cryptanalyst not only has access to the ciphertext and associated plaintext for several messages, but he also chooses the plaintext that gets encrypted. This is more powerful than a known-plaintext attack, because the cryptanalyst can choose specific plaintext blocks to encrypt, ones that might yield more information about the key. His job is to deduce the key (or keys) used to encrypt the messages or an algorithm to decrypt any new messages encrypted with the same key (or keys).

Given: $P_1, C_1 = E_k(P_1), P_2, C_2 = E_k(P_2), \dots, P_i, C_i = E_k(P_i)$, where the cryptanalyst gets to choose $P_1, P_2, \dots, P_i$

Deduce: Either k , or an algorithm to infer P_{i+1} from C_i

$+1 = E_k(P_{i+1})$

4. Adaptive-chosen-plaintext attack. This is a special case of a chosen-plaintext attack. Not only can the cryptanalyst choose the plaintext that is encrypted, but he can also modify his choice based on the results of previous encryption. In a chosen-plaintext attack, a cryptanalyst might just be able to choose one large block of plaintext to be encrypted; in an adaptive-chosen-plaintext attack he can choose a smaller block of plaintext and then choose another based on the results of the first, and so forth.

Previous	Table of Contents	Next
--------------------------	-----------------------------------	----------------------

[Products](#) | [Contact Us](#) | [About Us](#) | [Privacy](#) | [Ad Info](#) | [Home](#)

Use of this site is subject to certain [Terms & Conditions](#), [Copyright © 1996-2000 EarthWeb Inc.](#)

All rights reserved. Reproduction whole or in part in any form or medium without express written [permission](#) of EarthWeb is prohibited. Read EarthWeb's [privacy](#) statement.

☒ Brief
 ☐ Full

[Advanced Search](#)

[Search Tips](#)

 **BROWSE**
BY TOPIC

To access the contents, click the chapter and section titles.

Applied Cryptography, Second Edition: Protocols, Algorithms, and Source Code in C (cloth)

(Publisher: John Wiley & Sons, Inc.)

Author(s): Bruce Schneier

ISBN: 0471128457

Publication Date: 01/01/96

Bookmark It

Search this book:

[Previous](#)
[Table of Contents](#)
[Next](#)

There are at least three other types of cryptanalytic attack.

5. Chosen-ciphertext attack. The cryptanalyst can choose different ciphertexts to be decrypted and has access to the decrypted plaintext. For example, the cryptanalyst has access to a tamperproof box that does automatic decryption. His job is to deduce the key.

Given: $C_1, P_1 = D_k(C_1), C_2, P_2 = D_k(C_2), \dots, C_i, P_i = D_k(C_i)$

Deduce: k

This attack is primarily applicable to public-key algorithms and will be discussed in Section 19.3. A chosen-ciphertext attack is sometimes effective against a symmetric algorithm as well. (Sometimes a chosen-plaintext attack and a chosen-ciphertext attack are together known as a **chosen-text attack**.)

6. Chosen-key attack. This attack doesn't mean that the cryptanalyst can choose the key; it means that he has some knowledge about the relationship between different keys. It's strange and obscure, not very practical, and discussed in Section 12.4.

7. Rubber-hose cryptanalysis. The cryptanalyst threatens, blackmails, or tortures someone until they give him the key. Bribery is sometimes referred to as a **purchase-key attack**. These are all very powerful attacks and often the best way to break an algorithm.

Known-plaintext attacks and chosen-plaintext attacks are more common than you might think. It is not unheard-of for a cryptanalyst to get a plaintext message that has been encrypted or to bribe someone to encrypt a chosen message. You may not even have to bribe someone; if you give a message to an ambassador, you will probably find that it gets encrypted and sent back to his country for consideration. Many messages have standard beginnings and endings that might be known to the cryptanalyst. Encrypted source code is

especially vulnerable because of the regular appearance of keywords: #define, struct, else, return. Encrypted executable code has the same kinds of problems: functions, loop structures, and so on. Known-plaintext attacks (and even chosen-plaintext attacks) were successfully used against both the Germans and the Japanese during World War II. David Kahn's books [794,795,796] have historical examples of these kinds of attacks.

And don't forget Kerckhoffs's assumption: If the strength of your new cryptosystem relies on the fact that the attacker does not know the algorithm's inner workings, you're sunk. If you believe that keeping the algorithm's insides secret improves the security of your cryptosystem more than letting the academic community analyze it, you're wrong. And if you think that someone won't disassemble your code and reverse-engineer your algorithm, you're naïve. (In 1994 this happened with the RC4 algorithm—see Section 17.1.) The best algorithms we have are the ones that have been made public, have been attacked by the world's best cryptographers for years, and are still unbreakable. (The National Security Agency keeps their algorithms secret from outsiders, but they have the best cryptographers in the world working within their walls—you don't. Additionally, they discuss their algorithms with one another, relying on peer review to uncover any weaknesses in their work.)

Cryptanalysts don't always have access to the algorithms, as when the United States broke the Japanese diplomatic code PURPLE during World War II [794]—but they often do. If the algorithm is being used in a commercial security program, it is simply a matter of time and money to disassemble the program and recover the algorithm. If the algorithm is being used in a military communications system, it is simply a matter of time and money to buy (or steal) the equipment and reverse-engineer the algorithm.

Those who claim to have an unbreakable cipher simply because they can't break it are either geniuses or fools. Unfortunately, there are more of the latter in the world. Beware of people who extol the virtues of their algorithms, but refuse to make them public; trusting their algorithms is like trusting snake oil.

Good cryptographers rely on peer review to separate the good algorithms from the bad.

Security of Algorithms

Different algorithms offer different degrees of security; it depends on how hard they are to break. If the cost required to break an algorithm is greater than the value of the encrypted data, then you're probably safe. If the time required to break an algorithm is longer than the time the encrypted data must remain secret, then you're probably safe. If the amount of data encrypted with a single key is less than the amount of data necessary to break the algorithm, then you're probably safe.

I say "probably" because there is always a chance of new breakthroughs in cryptanalysis. On the other hand, the value of most data decreases over time. It is important that the value of the data always remain less than the cost to break the security protecting it.

Lars Knudsen classified these different categories of breaking an algorithm. In

decreasing order of severity [858]:

1. **Total break.** A cryptanalyst finds the key, K , such that $D_K(C) = P$.
2. **Global deduction.** A cryptanalyst finds an alternate algorithm, A , equivalent to $D_K(C)$, without knowing K .
3. **Instance (or local) deduction.** A cryptanalyst finds the plaintext of an intercepted ciphertext.
4. **Information deduction.** A cryptanalyst gains some information about the key or plaintext. This information could be a few bits of the key, some information about the form of the plaintext, and so forth.

An algorithm is **unconditionally secure** if, no matter how much ciphertext a cryptanalyst has, there is not enough information to recover the plaintext. In point of fact, only a one-time pad (see Section 1.5) is unbreakable given infinite resources. All other cryptosystems are breakable in a ciphertext-only attack, simply by trying every possible key one by one and checking whether the resulting plaintext is meaningful. This is called a **brute-force** attack (see Section 7.1).

Cryptography is more concerned with cryptosystems that are computationally infeasible to break. An algorithm is considered **computationally secure** (sometimes called strong) if it cannot be broken with available resources, either current or future. Exactly what constitutes “available resources” is open to interpretation.

You can measure the complexity (see Section 11.1) of an attack in different ways:

1. **Data complexity.** The amount of data needed as input to the attack.
2. **Processing complexity.** The time needed to perform the attack. This is often called the **work factor**.
3. **Storage requirements.** The amount of memory needed to do the attack.

As a rule of thumb, the complexity of an attack is taken to be the minimum of these three factors. Some attacks involve trading off the three complexities: A faster attack might be possible at the expense of a greater storage requirement.

Complexities are expressed as orders of magnitude. If an algorithm has a processing complexity of 2^{128} , then 2^{128} operations are required to break the algorithm. (These operations may be complex and time-consuming.) Still, if you assume that you have enough computing speed to perform a million operations every second and you set a million parallel processors against the task, it will still take over 10^{19} years to recover the key. That’s a billion times the age of the universe.

Previous	Table of Contents	Next
--------------------------	-----------------------------------	----------------------

[Products](#) | [Contact Us](#) | [About Us](#) | [Privacy](#) | [Ad Info](#) | [Home](#)

Use of this site is subject to certain [Terms & Conditions](#), [Copyright © 1996-2000 EarthWeb Inc.](#)

All rights reserved. Reproduction whole or in part in any form or medium without express written [permission](#) of EarthWeb is prohibited. Read EarthWeb's [privacy](#) statement.

 SEARCH
ITKNOWLEDGE

☒ Brief ☐ Full

[Advanced Search](#)

[Search Tips](#)

 BROWSE
BY TOPIC

To access the contents, click the chapter and section titles.

Applied Cryptography, Second Edition: Protocols, Algorithms, and Source Code in C (cloth)

(Publisher: John Wiley & Sons, Inc.)

Author(s): Bruce Schneier

ISBN: 0471128457

Publication Date: 01/01/96

Bookmark It

Search this book:

[Previous](#)
[Table of Contents](#)
[Next](#)

While the complexity of an attack is constant (until some cryptanalyst finds a better attack, of course), computing power is anything but. There have been phenomenal advances in computing power during the last half-century and there is no reason to think this trend won't continue. Many cryptanalytic attacks are perfect for parallel machines: The task can be broken down into billions of tiny pieces and none of the processors need to interact with each other. Pronouncing an algorithm secure simply because it is infeasible to break, given current technology, is dicey at best. Good cryptosystems are designed to be infeasible to break with the computing power that is expected to evolve many years in the future.

Historical Terms

Historically, a **code** refers to a cryptosystem that deals with linguistic units: words, phrases, sentences, and so forth. For example, the word "OCELOT" might be the ciphertext for the entire phrase "TURN LEFT 90 DEGREES," the word "LOLLIPOP" might be the ciphertext for "TURN RIGHT 90 DEGREES," and the words "BENT EAR" might be the ciphertext for "HOWITZER." Codes of this type are not discussed in this book; see [794,795]. Codes are only useful for specialized circumstances. Ciphers are useful for any circumstance. If your code has no entry for "ANTEATERS," then you can't say it. You can say anything with a cipher.

1.2 Steganography

Steganography serves to hide secret messages in other messages, such that the secret's very existence is concealed. Generally the sender writes an innocuous message and then conceals a secret message on the same piece of paper. Historical tricks include invisible inks, tiny pin punctures on selected characters, minute differences between handwritten characters, pencil marks

on typewritten characters, grilles which cover most of the message except for a few characters, and so on.

More recently, people are hiding secret messages in graphic images. Replace the least significant bit of each byte of the image with the bits of the message. The graphical image won't change appreciably—most graphics standards specify more gradations of color than the human eye can notice—and the message can be stripped out at the receiving end. You can store a 64-kilobyte message in a 1024×1024 grey-scale picture this way. Several public-domain programs do this sort of thing.

Peter Wayner's **mimic functions** obfuscate messages. These functions modify a message so that its statistical profile resembles that of something else: the classifieds section of *The New York Times*, a play by Shakespeare, or a newsgroup on the Internet [1584,1585]. This type of steganography won't fool a person, but it might fool some big computers scanning the Internet for interesting messages.

1.3 Substitution Ciphers and Transposition Ciphers

Before computers, cryptography consisted of character-based algorithms. Different cryptographic algorithms either substituted characters for one another or transposed characters with one another. The better algorithms did both, many times each.

Things are more complex these days, but the philosophy remains the same. The primary change is that algorithms work on bits instead of characters. This is actually just a change in the alphabet size: from 26 elements to two elements. Most good cryptographic algorithms still combine elements of substitution and transposition.

Substitution Ciphers

A **substitution cipher** is one in which each character in the plaintext is substituted for another character in the ciphertext. The receiver inverts the substitution on the ciphertext to recover the plaintext.

In classical cryptography, there are four types of substitution ciphers:

- A **simple substitution cipher**, or **monoalphabetic cipher**, is one in which each character of the plaintext is replaced with a corresponding character of ciphertext. The cryptograms in newspapers are simple substitution ciphers.

- A **homophonic substitution cipher** is like a simple substitution cryptosystem, except a single character of plaintext can map to one of several characters of ciphertext. For example, "A" could correspond to either 5, 13, 25, or 56, "B" could correspond to either 7, 19, 31, or 42, and so on.

- A **polygram substitution cipher** is one in which blocks of characters are encrypted in groups. For example, "ABA" could correspond to "RTLQ," "ABB" could correspond to "SLL," and so on.

- A **polyalphabetic substitution cipher** is made up of multiple simple

substitution ciphers. For example, there might be five different simple substitution ciphers used; the particular one used changes with the position of each character of the plaintext.

The famous **Caesar Cipher**, in which each plaintext character is replaced by the character three to the right modulo 26 (“A” is replaced by “D,” “B” is replaced by “E,” ..., “W” is replaced by “Z,” “X” is replaced by “A,” “Y” is replaced by “B,” and “Z” is replaced by “C”) is a simple substitution cipher. It’s actually even simpler, because the ciphertext alphabet is a rotation of the plaintext alphabet and not an arbitrary permutation.

ROT13 is a simple encryption program commonly found on UNIX systems; it is also a simple substitution cipher. In this cipher, “A” is replaced by “N,” “B” is replaced by “O,” and so on. Every letter is rotated 13 places.

Encrypting a file twice with ROT13 restores the original file.

$$P = \text{ROT13}(\text{ROT13}(P))$$

ROT13 is not intended for security; it is often used in Usenet posts to hide potentially offensive text, to avoid giving away the solution to a puzzle, and so forth.

Simple substitution ciphers can be easily broken because the cipher does not hide the underlying frequencies of the different letters of the plaintext. All it takes is about 25 English characters before a good cryptanalyst can reconstruct the plaintext [1434]. An algorithm for solving these sorts of ciphers can be found in [578,587,1600,78,1475,1236,880]. A good computer algorithm is [703].

Homophonic substitution ciphers were used as early as 1401 by the Duchy of Mantua [794]. They are much more complicated to break than simple substitution ciphers, but still do not obscure all of the statistical properties of the plaintext language. With a known-plaintext attack, the ciphers are trivial to break. A ciphertext-only attack is harder, but only takes a few seconds on a computer. Details are in [1261].

Polygram substitution ciphers are ciphers in which groups of letters are encrypted together. The Playfair cipher, invented in 1854, was used by the British during World War I [794]. It encrypts pairs of letters together. Its cryptanalysis is discussed in [587,1475,880]. The Hill cipher is another example of a polygram substitution cipher [732]. Sometimes you see Huffman coding used as a cipher; this is an insecure polygram substitution cipher.

Polyalphabetic substitution ciphers were invented by Leon Battista in 1568 [794]. They were used by the Union army during the American Civil War. Despite the fact that they can be broken easily [819,577,587,794] (especially with the help of computers), many commercial computer security products use ciphers of this form [1387,1390,1502]. (Details on how to break this encryption scheme, as used in WordPerfect, can be found in [135,139].) The Vigenère cipher, first published in 1586, and the Beaufort cipher are also examples of polyalphabetic substitution ciphers.

Polyalphabetic substitution ciphers have multiple one-letter keys, each of

which is used to encrypt one letter of the plaintext. The first key encrypts the first letter of the plaintext, the second key encrypts the second letter of the plaintext, and so on. After all the keys are used, the keys are recycled. If there were 20 one-letter keys, then every twentieth letter would be encrypted with the same key. This is called the **period** of the cipher. In classical cryptography, ciphers with longer periods were significantly harder to break than ciphers with short periods. There are computer techniques that can easily break substitution ciphers with very long periods.

A **running-key cipher**—sometimes called a book cipher—in which one text is used to encrypt another text, is another example of this sort of cipher. Even though this cipher has a period the length of the text, it can also be broken easily [576,794].

Previous	Table of Contents	Next
--------------------------	-----------------------------------	----------------------

[Products](#) | [Contact Us](#) | [About Us](#) | [Privacy](#) | [Ad Info](#) | [Home](#)

Use of this site is subject to certain [Terms & Conditions](#), [Copyright © 1996-2000 EarthWeb Inc.](#)

All rights reserved. Reproduction whole or in part in any form or medium without express written [permission](#) of EarthWeb is prohibited. Read EarthWeb's [privacy](#) statement.

 SEARCH
ITKNOWLEDGE

☒ Brief ☐ Full

[Advanced Search](#)

[Search Tips](#)

 BROWSE
BY TOPIC

To access the contents, click the chapter and section titles.

Applied Cryptography, Second Edition: Protocols, Algorithms, and Source Code in C (cloth)

(Publisher: John Wiley & Sons, Inc.)

Author(s): Bruce Schneier

ISBN: 0471128457

Publication Date: 01/01/96

Bookmark It

Search this book:

[Previous](#)
[Table of Contents](#)
[Next](#)

Transposition Ciphers

In a **transposition cipher** the plaintext remains the same, but the order of characters is shuffled around. In a **simple columnar transposition cipher**, the plaintext is written horizontally onto a piece of graph paper of fixed width and the ciphertext is read off vertically (see Figure 1.4). Decryption is a matter of writing the ciphertext vertically onto a piece of graph paper of identical width and then reading the plaintext off horizontally.

Cryptanalysis of these ciphers is discussed in [587,1475]. Since the letters of the ciphertext are the same as those of the plaintext, a frequency analysis on the ciphertext would reveal that each letter has approximately the same likelihood as in English. This gives a very good clue to a cryptanalyst, who can then use a variety of techniques to determine the right ordering of the letters to obtain the plaintext. Putting the ciphertext through a second transposition cipher greatly enhances security. There are even more complicated transposition ciphers, but computers can break almost all of them.

The German ADFGVX cipher, used during World War I, is a transposition cipher combined with a simple substitution. It was a very complex algorithm for its day but was broken by Georges Painvin, a French cryptanalyst [794].

Although many modern algorithms use transposition, it is troublesome because it requires a lot of memory and sometimes requires messages to be only certain lengths. Substitution is far more common.

Rotor Machines

In the 1920s, various mechanical encryption devices were invented to automate the process of encryption. Most were based on the concept of a **rotor**, a mechanical wheel wired to perform a general substitution.

A **rotor machine** has a keyboard and a series of rotors, and implements a version of the Vigenère cipher. Each rotor is an arbitrary permutation of the alphabet, has 26 positions, and performs a simple substitution. For example, a rotor might be wired to substitute “F” for “A,” “U” for “B,” “L” for “C,” and so on. And the output pins of one rotor are connected to the input pins of the next.



Figure 1.4 *Columnar transposition cipher.*

For example, in a 4-rotor machine the first rotor might substitute “F” for “A,” the second might substitute “Y” for “F,” the third might substitute “E” for “Y,” and the fourth might substitute “C” for “E”; “C” would be the output ciphertext. Then some of the rotors shift, so next time the substitutions will be different.

It is the combination of several rotors and the gears moving them that makes the machine secure. Because the rotors all move at different rates, the period for an n -rotor machine is 26^n . Some rotor machines can also have a different number of positions on each rotor, further frustrating cryptanalysis.

The best-known rotor device is the Enigma. The Enigma was used by the Germans during World War II. The idea was invented by Arthur Scherbius and Arvid Gerhard Damm in Europe. It was patented in the United States by Arthur Scherbius [1383]. The Germans beefed up the basic design considerably for wartime use.

The German Enigma had three rotors, chosen from a set of five, a plugboard that slightly permuted the plaintext, and a reflecting rotor that caused each rotor to operate on each plaintext letter twice. As complicated as the Enigma was, it was broken during World War II. First, a team of Polish cryptographers broke the German Enigma and explained their attack to the British. The Germans modified their Enigma as the war progressed, and the British continued to cryptanalyze the new versions. For explanations of how rotor ciphers work and how they were broken, see [794,86,448,498,446,880,1315,1587,690]. Two fascinating accounts of how the Enigma was broken are [735,796].

Further Reading

This is not a book about classical cryptography, so I will not dwell further on these subjects. Two excellent precomputer cryptology books are [587,1475]; [448] presents some modern cryptanalysis of cipher machines. Dorothy Denning discusses many of these ciphers in [456] and [880] has some fairly complex mathematical analysis of the same ciphers. Another older cryptography text, which discusses analog cryptography, is [99]. An article that presents a good overview of the subject is [579]. David Kahn’s historical cryptography books are also excellent [794,795,796].

1.4 Simple XOR

XOR is exclusive-or operation: ‘^’ in C or • in mathematical notation. It’s a standard operation on bits:

$$0 \bullet 0 = 0$$

$$0 \bullet 1 = 1$$

$$1 \bullet 0 = 1$$

$$1 \bullet 1 = 0$$

Also note that:

$$a \bullet a = 0$$

$$a \bullet b \bullet b = a$$

The simple-XOR algorithm is really an embarrassment; it's nothing more than a Vigenère polyalphabetic cipher. It's here only because of its prevalence in commercial software packages, at least those in the MS-DOS and Macintosh worlds [1502,1387]. Unfortunately, if a software security program proclaims that it has a "proprietary" encryption algorithm—significantly faster than DES—the odds are that it is some variant of this.

```
/* Usage:  crypto key input_file output_file */

void main (int argc, char *argv[])
{
    FILE *fi, *fo;
    char *cp;
    int c;

    if ((cp = argv[1]) && *cp!='\0') {
        if ((fi = fopen(argv[2], "rb")) != NULL) {
            if ((fo = fopen(argv[3], "wb")) != NULL) {
                while ((c = getc(fi)) != EOF) {
                    if (!*cp) cp = argv[1];
                    c ^= *(cp++);
                    putc(c, fo);
                }
                fclose(fo);
            }
            fclose(fi);
        }
    }
}
```

This is a symmetric algorithm. The plaintext is being XORed with a keyword to generate the ciphertext. Since XORing the same value twice restores the original, encryption and decryption use exactly the same program:

$$P \bullet K = C$$

$$C \bullet K = P$$

There's no real security here. This kind of encryption is trivial to break, even without computers [587,1475]. It will only take a few seconds with a computer.

Assume the plaintext is English. Furthermore, assume the key length is any small number of bytes. Here's how to break it:

1. Discover the length of the key by a procedure known as **counting coincidences** [577]. XOR the ciphertext against itself shifted various numbers of bytes, and count those bytes that are equal. If the displacement is a multiple of the key length, then something over 6 percent of the bytes will be equal. If it is not, then less than 0.4 percent will be equal (assuming a random key encrypting normal ASCII text; other plaintext will have different numbers). This is called the **index of coincidence**. The smallest displacement that indicates a multiple of the key length is the length of the key.
2. Shift the ciphertext by that length and XOR it with itself. This removes the key and leaves you with plaintext XORed with the plaintext shifted the length of the key. Since English has 1.3 bits of real information per byte (see Section 11.1), there is plenty of redundancy for determining a unique decryption.

Despite this, the list of software vendors that tout this toy algorithm as being “almost as secure as DES” is staggering [1387]. It is the algorithm (with a 160-bit repeated “key”) that the NSA finally allowed the U.S. digital cellular phone industry to use for voice privacy. An XOR might keep your kid sister from reading your files, but it won’t stop a cryptanalyst for more than a few minutes.

1.5 One-Time Pads

Believe it or not, there is a perfect encryption scheme. It’s called a **one-time pad**, and was invented in 1917 by Major Joseph Mauborgne and AT&T’s Gilbert Vernam [794]. (Actually, a one-time pad is a special case of a threshold scheme; see Section 3.7.) Classically, a one-time pad is nothing more than a large nonrepeating set of truly random key letters, written on sheets of paper, and glued together in a pad. In its original form, it was a one-time tape for teletypewriters. The sender uses each key letter on the pad to encrypt exactly one plaintext character. Encryption is the addition modulo 26 of the plaintext character and the one-time pad key character.

Each key letter is used exactly once, for only one message. The sender encrypts the message and then destroys the used pages of the pad or used section of the tape. The receiver has an identical pad and uses each key on the pad, in turn, to decrypt each letter of the ciphertext. The receiver destroys the same pad pages or tape section after decrypting the message. New message—new key letters. For example, if the message is:

ONETIMEPAD

and the key sequence from the pad is

TBFRGFARFM

then the ciphertext is

IPKLPSFHGQ

because

$O + T \bmod 26 = I$
 $N + B \bmod 26 = P$
 $E + F \bmod 26 = K$
etc.

Previous	Table of Contents	Next
--------------------------	-----------------------------------	----------------------

[Products](#) | [Contact Us](#) | [About Us](#) | [Privacy](#) | [Ad Info](#) | [Home](#)

Use of this site is subject to certain [Terms & Conditions](#), [Copyright © 1996-2000 EarthWeb Inc.](#)

All rights reserved. Reproduction whole or in part in any form or medium without express written [permission](#) of EarthWeb is prohibited. Read EarthWeb's [privacy](#) statement.

SEARCH
ITKNOWLEDGE

Keyword

☒ Brief ☐ Full

[Advanced Search](#)

[Search Tips](#)

BROWSE
BY TOPIC

To access the contents, click the chapter and section titles.

Applied Cryptography, Second Edition: Protocols, Algorithms, and Source Code in C (cloth)

(Publisher: John Wiley & Sons, Inc.)

Author(s): Bruce Schneier

ISBN: 0471128457

Publication Date: 01/01/96

[Bookmark It](#)

Search this book:

[Previous](#) [Table of Contents](#) [Next](#)

Assuming an eavesdropper can't get access to the one-time pad used to encrypt the message, this scheme is perfectly secure. A given ciphertext message is equally likely to correspond to any possible plaintext message of equal size.

Since every key sequence is equally likely (remember, the key letters are generated randomly), an adversary has no information with which to cryptanalyze the ciphertext. The key sequence could just as likely be:

POYYAEAAZX

which would decrypt to:

SALMONEGGS

or

BXFGBMTMXM

which would decrypt to:

GREENFLUID

This point bears repeating: Since every plaintext message is equally possible, there is no way for the cryptanalyst to determine which plaintext message is the correct one. A random key sequence added to a nonrandom plaintext message produces a completely random ciphertext message and no amount of computing power can change that.

The caveat, and this is a big one, is that the key letters have to be generated randomly. Any attacks against this scheme will be against the method used to generate the key letters. Using a pseudo-random number generator doesn't

count; they always have nonrandom properties. If you use a real random source—this is much harder than it might first appear, see Section 17.14—it’s secure.

The other important point is that you can never use the key sequence again, ever. Even if you use a multiple-gigabyte pad, if a cryptanalyst has multiple ciphertexts whose keys overlap, he can reconstruct the plaintext. He slides each pair of ciphertexts against each other and counts the number of matches at each position. If they are aligned right, the proportion of matches jumps suddenly—the exact percentages depend on the plaintext language. From this point cryptanalysis is easy. It’s like the index of coincidence, but with just two “periods” to compare [904]. Don’t do it.

The idea of a one-time pad can be easily extended to binary data. Instead of a one-time pad consisting of letters, use a one-time pad of bits. Instead of adding the plaintext to the one-time pad, use an XOR. To decrypt, XOR the ciphertext with the same one-time pad. Everything else remains the same and the security is just as perfect.

This all sounds good, but there are a few problems. Since the key bits must be random and can never be used again, the length of the key sequence must be equal to the length of the message. A one-time pad might be suitable for a few short messages, but it will never work for a 1.544 Mbps communications channel. You can store 650 megabytes worth of random bits on a CD-ROM, but there are problems. First, you want exactly two copies of the random bits, but CD-ROMs are economical only for large quantities. And second, you want to be able to destroy the bits already used. CD-ROM has no erase facilities except for physically destroying the entire disk. Digital tape is a much better medium for this sort of thing.

Even if you solve the key distribution and storage problem, you have to make sure the sender and receiver are perfectly synchronized. If the receiver is off by a bit (or if some bits are dropped during the transmission), the message won’t make any sense. On the other hand, if some bits are altered during transmission (without any bits being added or removed—something far more likely to happen due to random noise), only those bits will be decrypted incorrectly. But on the other hand, a one-time pad provides no authenticity.

One-time pads have applications in today’s world, primarily for ultra-secure low-bandwidth channels. The hotline between the United States and the former Soviet Union was (is it still active?) rumored to be encrypted with a one-time pad. Many Soviet spy messages to agents were encrypted using one-time pads. These messages are still secure today and will remain that way forever. It doesn’t matter how long the supercomputers work on the problem. Even after the aliens from Andromeda land with their massive spaceships and undreamed-of computing power, they will not be able to read the Soviet spy messages encrypted with one-time pads (unless they can also go back in time and get the one-time pads).

1.6 Computer Algorithms

There are many cryptographic algorithms. These are three of the most

common:

- DES (Data Encryption Standard) is the most popular computer encryption algorithm. DES is a U.S. and international standard. It is a symmetric algorithm; the same key is used for encryption and decryption.
- RSA (named for its creators—Rivest, Shamir, and Adleman) is the most popular public-key algorithm. It can be used for both encryption and digital signatures.
- DSA (Digital Signature Algorithm, used as part of the Digital Signature Standard) is another public-key algorithm. It cannot be used for encryption, but only for digital signatures.

These are the kinds of stuff this book is about.

1.7 Large Numbers

Throughout this book, I use various large numbers to describe different things in cryptography. Because it is so easy to lose sight of these numbers and what they signify, Table 1.1 gives physical analogues for some of them.

These numbers are order-of-magnitude estimates, and have been culled from a variety of sources. Many of the astrophysics numbers are explained in Freeman

TABLE 1.1
Large Numbers

Physical Analogue	Number
Odds of being killed by lightning (per day)	1 in 9 billion (2^{33})
Odds of winning the top prize in a U.S. state lottery	1 in 4,000,000 (2^{22})
Odds of winning the top prize in a U.S. state lottery and being killed by lightning in the same day	1 in 2^{55}
Odds of drowning (in the U.S. per year)	1 in 59,000 (2^{16})
Odds of being killed in an automobile accident(in the U.S. in 1993)	1 in 6100 (2^{13})
Odds of being killed in an automobile accident(in the U.S. per lifetime)	1 in 88 (2^7)
Time until the next ice age	14,000 (2^{14}) years
Time until the sun goes nova	10^9 (2^{30}) years
Age of the planet	10^9 (2^{30}) years
Age of the Universe	10^{10} (2^{34}) years
Number of atoms in the planet	10^{51} (2^{170})
Number of atoms in the sun	10^{57} (2^{190})
Number of atoms in the galaxy	10^{67} (2^{223})
Number of atoms in the Universe (dark matter excluded)	10^{77} (2^{265})
Volume of the Universe	10^{84} (2^{280}) cm^3

If the Universe is Closed:

Total lifetime of the Universe	10^{11} (2^{37}) years
	10^{18} (2^{61}) seconds

If the Universe is Open:

Time until low-mass stars cool off	10^{14} (2^{47}) years
Time until planets detach from stars	10^{15} (2^{50}) years
Time until stars detach from galaxies	10^{19} (2^{64}) years
Time until orbits decay by gravitational radiation	10^{20} (2^{67}) years
Time until black holes decay by the Hawking process	10^{64} (2^{213}) years
Time until all matter is liquid at zero temperature	10^{65} (2^{216}) years
Time until all matter decays to iron	10^{1026} years
Time until all matter collapses to black holes	10^{1076} years

Dyson’s paper, “Time Without End: Physics and Biology in an Open Universe,” in *Reviews of Modern Physics*, v. 52, n. 3, July 1979, pp. 447–460. Automobile accident deaths are calculated from the Department of Transportation’s statistic of 163 deaths per million people in 1993 and an average lifespan of 69.7 years.

Previous	Table of Contents	Next
--------------------------	-----------------------------------	----------------------

[Products](#) | [Contact Us](#) | [About Us](#) | [Privacy](#) | [Ad Info](#) | [Home](#)

Use of this site is subject to certain [Terms & Conditions](#), [Copyright © 1996-2000 EarthWeb Inc.](#)
All rights reserved. Reproduction whole or in part in any form or medium without express written [permission](#) of EarthWeb is prohibited. Read EarthWeb's [privacy](#) statement.

 SEARCH
ITKNOWLEDGE

☒ Brief ☐ Full

[Advanced Search](#)

[Search Tips](#)

 BROWSE
BY TOPIC

To access the contents, click the chapter and section titles.

Applied Cryptography, Second Edition: Protocols, Algorithms, and Source Code in C (cloth)

(Publisher: John Wiley & Sons, Inc.)

Author(s): Bruce Schneier

ISBN: 0471128457

Publication Date: 01/01/96

Bookmark It

Search this book:

[Previous](#)
[Table of Contents](#)
[Next](#)

Part I

Cryptographic protocols

Chapter 2

Protocol Building Blocks

2.1 Introduction to Protocols

The whole point of cryptography is to solve problems. (Actually, that's the whole point of computers—something many people tend to forget.) Cryptography solves problems that involve secrecy, authentication, integrity, and dishonest people. You can learn all about cryptographic algorithms and techniques, but these are academic unless they can solve a problem. This is why we are going to look at protocols first.

A **protocol** is a series of steps, involving two or more parties, designed to accomplish a task. This is an important definition. A “series of steps” means that the protocol has a sequence, from start to finish. Every step must be executed in turn, and no step can be taken before the previous step is finished. “Involving two or more parties” means that at least two people are required to complete the protocol; one person alone does not make a protocol. A person alone can perform a series of steps to accomplish a task (like baking a cake), but this is not a protocol. (Someone else must eat the cake to make it a protocol.) Finally, “designed to accomplish a task” means that the protocol must achieve something. Something that looks like a protocol but does not accomplish a task is not a protocol—it's a waste of time.

Protocols have other characteristics as well:

- Everyone involved in the protocol must know the protocol and all of the steps to follow in advance.
- Everyone involved in the protocol must agree to follow it.
- The protocol must be unambiguous; each step must be well defined and there must be no chance of a misunderstanding.
- The protocol must be complete; there must be a specified action for every possible situation.

The protocols in this book are organized as a series of steps. Execution of the protocol proceeds linearly through the steps, unless there are instructions to branch to another step. Each step involves at least one of two things: computations by one or more of the parties, or messages sent among the parties.

A **cryptographic protocol** is a protocol that uses cryptography. The parties can be friends and trust each other implicitly or they can be adversaries and not trust one another to give the correct time of day. A cryptographic protocol involves some cryptographic algorithm, but generally the goal of the protocol is something beyond simple secrecy. The parties participating in the protocol might want to share parts of their secrets to compute a value, jointly generate a random sequence, convince one another of their identity, or simultaneously sign a contract. The whole point of using cryptography in a protocol is to prevent or detect eavesdropping and cheating. If you have never seen these protocols before, they will radically change your ideas of what mutually distrustful parties can accomplish over a computer network. In general, this can be stated as:

- It should not be possible to do more or learn more than what is specified in the protocol.

This is a lot harder than it looks. In the next few chapters I discuss a lot of protocols. In some of them it is possible for one of the participants to cheat the other. In others, it is possible for an eavesdropper to subvert the protocol or learn secret information. Some protocols fail because the designers weren't thorough enough in their requirements definitions. Others fail because their designers weren't thorough enough in their analysis. Like algorithms, it is much easier to prove insecurity than it is to prove security.

The Purpose of Protocols

In daily life, there are informal protocols for almost everything: ordering goods over the telephone, playing poker, voting in an election. No one thinks much about these protocols; they have evolved over time, everyone knows how to use them, and they work reasonably well.

These days, more and more human interaction takes place over computer networks instead of face-to-face. Computers need formal protocols to do the same things that people do without thinking. If you moved from one state to another and found a voting booth that looked completely different from the ones you were used to, you could easily adapt. Computers are not nearly so flexible.

Many face-to-face protocols rely on people's presence to ensure fairness and security. Would you send a stranger a pile of cash to buy groceries for you? Would you play poker with someone if you couldn't see him shuffle and deal? Would you mail the government your secret ballot without some assurance of anonymity?

It is naïve to assume that people on computer networks are honest. It is naïve to assume that the managers of computer networks are honest. It is even naïve to assume that the designers of computer networks are honest. Most are, but the dishonest few can do a lot of damage. By formalizing protocols, we can examine ways in which dishonest parties can subvert them. Then we can develop protocols that are immune to that subversion.

In addition to formalizing behavior, protocols abstract the process of accomplishing a task from the mechanism by which the task is accomplished. A communications protocol is the same whether implemented on PCs or VAXs. We can examine the protocol without getting bogged down in the implementation details. When we are convinced we have a good protocol, we can implement it in everything from computers to telephones to intelligent muffin toasters.

The Players

To help demonstrate protocols, I have enlisted the aid of several people (see Table 2.1). Alice and Bob are the first two. They will perform all general two-person protocols. As a rule, Alice will initiate all protocols and Bob will respond. If the protocol requires a third or fourth person, Carol and Dave will perform those roles. Other actors will play specialized supporting roles; they will be introduced later.

Arbitrated Protocols

An **arbitrator** is a disinterested third party trusted to complete a protocol (see Figure 2.1a). Disinterested means that the arbitrator has no vested interest in the protocol and no particular allegiance to any of the parties involved. Trusted means that all people involved in the protocol accept what he says as true, what he does as correct, and that he will complete his part of the protocol. Arbitrators can help complete protocols between two mutually distrustful parties.

Previous	Table of Contents	Next
--------------------------	-----------------------------------	----------------------

[Products](#) | [Contact Us](#) | [About Us](#) | [Privacy](#) | [Ad Info](#) | [Home](#)

Use of this site is subject to certain [Terms & Conditions](#), [Copyright © 1996-2000 EarthWeb Inc.](#)

All rights reserved. Reproduction whole or in part in any form or medium without express written [permission](#) of EarthWeb is prohibited. Read EarthWeb's [privacy](#) statement.

SEARCH
ITKNOWLEDGE

Keyword

☒ Brief
 ☐ Full

[Advanced Search](#)

[Search Tips](#)

BROWSE
BY TOPIC

To access the contents, click the chapter and section titles.

Applied Cryptography, Second Edition: Protocols, Algorithms, and Source Code in C (cloth)
(Publisher: John Wiley & Sons, Inc.)
Author(s): Bruce Schneier
ISBN: 0471128457
Publication Date: 01/01/96

Bookmark It

Search this book:

[Previous](#)

[Table of Contents](#)

[Next](#)

In the real world, lawyers are often used as arbitrators. For example, Alice is selling a car to Bob, a stranger. Bob wants to pay by check, but Alice has no way of knowing if the check is good. Alice wants the check to clear before she turns the title over to Bob. Bob, who doesn't trust Alice any more than she trusts him, doesn't want to hand over a check without receiving a title.

TABLE 2.1
Dramatis Personae

Alice	First participant in all the protocols
Bob	Second participant in all the protocols
Carol	Participant in the three- and four-party protocols
Dave	Participant in the four-party protocols
Eve	Eavesdropper
Mallory	Malicious active attacker
Trent	Trusted arbitrator
Walter	Warden; he'll be guarding Alice and Bob in some protocols
Peggy	Prover
Victor	Verifier

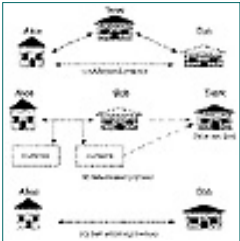


Figure 2.1 Types of protocols.

Enter a lawyer trusted by both. With his help, Alice and Bob can use the following protocol to ensure that neither cheats the other:

- (1) Alice gives the title to the lawyer.
- (2) Bob gives the check to Alice.
- (3) Alice deposits the check.
- (4) After waiting a specified time period for the check to clear, the lawyer gives the title to Bob. If the check does not clear within the specified time period, Alice shows proof of this to the lawyer and the lawyer returns the title to Alice.

In this protocol, Alice trusts the lawyer not to give Bob the title unless the check has cleared, and to give it back to her if the check does not clear. Bob trusts the lawyer to hold the title until the check clears, and to give it to him once it does. The lawyer doesn't care if the check clears. He will do his part of the protocol in either case, because he will be paid in either case.

In the example, the lawyer is playing the part of an escrow agent. Lawyers also act as arbitrators for wills and sometimes for contract negotiations. The various stock exchanges act as arbitrators between buyers and sellers.

Bankers also arbitrate protocols. Bob can use a certified check to buy a car from Alice:

- (1) Bob writes a check and gives it to the bank.
- (2) After putting enough of Bob's money on hold to cover the check, the bank certifies the check and gives it back to Bob.
- (3) Alice gives the title to Bob and Bob gives the certified check to Alice.
- (4) Alice deposits the check.

This protocol works because Alice trusts the banker's certification. Alice trusts the bank to hold Bob's money for her, and not to use it to finance shaky real estate operations in mosquito-infested countries.

A notary public is another arbitrator. When Bob receives a notarized document from Alice, he is convinced that Alice signed the document voluntarily and with her own hand. The notary can, if necessary, stand up in court and attest to that fact.

The concept of an arbitrator is as old as society. There have always been people—rulers, priests, and so on—who have the authority to act fairly. Arbitrators have a certain social role and position in our society; betraying the public trust would jeopardize that. Lawyers who play games with escrow accounts face almost-certain disbarment, for example. This picture of trust doesn't always exist in the real world, but it's the ideal.

This ideal can translate to the computer world, but there are several problems with computer arbitrators:

- It is easier to find and trust a neutral third party if you know who the party is and can see his face. Two parties suspicious of each other are also likely to be suspicious of a faceless arbitrator somewhere else on the network.

- The computer network must bear the cost of maintaining an arbitrator. We all know what lawyers charge; who wants to bear that kind of network overhead?
- There is a delay inherent in any arbitrated protocol.
- The arbitrator must deal with every transaction; he is a bottleneck in large-scale implementations of any protocol. Increasing the number of arbitrators in the implementation can mitigate this problem, but that increases the cost.
- Since everyone on the network must trust the arbitrator, he represents a vulnerable point for anyone trying to subvert the network.

Even so, arbitrators still have a role to play. In protocols using a trusted arbitrator, the part will be played by Trent.

Adjudicated Protocols

Because of the high cost of hiring arbitrators, arbitrated protocols can be subdivided into two lower-level **subprotocols**. One is a nonarbitrated subprotocol, executed every time parties want to complete the protocol. The other is an arbitrated subprotocol, executed only in exceptional circumstances—when there is a dispute. This special type of arbitrator is called an **adjudicator** (see Figure 2.1b).

An adjudicator is also a disinterested and trusted third party. Unlike an arbitrator, he is not directly involved in every protocol. The adjudicator is called in only to determine whether a protocol was performed fairly.

Judges are professional adjudicators. Unlike a notary public, a judge is brought in only if there is a dispute. Alice and Bob can enter into a contract without a judge. A judge never sees the contract until one of them hauls the other into court.

This contract-signing protocol can be formalized in this way:

Nonarbitrated subprotocol (executed every time):

- (1) Alice and Bob negotiate the terms of the contract.
- (2) Alice signs the contract.
- (3) Bob signs the contract.

Adjudicated subprotocol (executed only in case of a dispute):

- (4) Alice and Bob appear before a judge.
- (5) Alice presents her evidence.
- (6) Bob presents his evidence.
- (7) The judge rules on the evidence.

[Products](#) | [Contact Us](#) | [About Us](#) | [Privacy](#) | [Ad Info](#) | [Home](#)

Use of this site is subject to certain [Terms & Conditions](#), [Copyright © 1996-2000 EarthWeb Inc.](#)

All rights reserved. Reproduction whole or in part in any form or medium without express written [permission](#) of EarthWeb is prohibited. Read EarthWeb's [privacy](#) statement.

Go!

Keyword

☒ Brief ☐ Full

[Advanced Search](#)

[Search Tips](#)

BROWSE
BY TOPIC

To access the contents, click the chapter and section titles.

Applied Cryptography, Second Edition: Protocols, Algorithms, and Source Code in C (cloth)

(Publisher: John Wiley & Sons, Inc.)

Author(s): Bruce Schneier

ISBN: 0471128457

Publication Date: 01/01/96

[Bookmark It](#)

Search this book:

Go!

[Previous](#) [Table of Contents](#) [Next](#)

The difference between an adjudicator and an arbitrator (as used in this book) is that the adjudicator is not always necessary. In a dispute, a judge is called in to adjudicate. If there is no dispute, using a judge is unnecessary.

There are adjudicated computer protocols. These protocols rely on the parties to be honest; but if someone suspects cheating, a body of data exists so that a trusted third party could determine if someone cheated. In a good adjudicated protocol, the adjudicator could also determine the cheater's identity. Instead of preventing cheating, adjudicated protocols detect cheating. The inevitability of detection acts as a preventive and discourages cheating.

Self-Enforcing Protocols

A **self-enforcing protocol** is the best type of protocol. The protocol itself guarantees fairness (see Figure 2.1c). No arbitrator is required to complete the protocol. No adjudicator is required to resolve disputes. The protocol is constructed so that there cannot be any disputes. If one of the parties tries to cheat, the other party immediately detects the cheating and the protocol stops. Whatever the cheating party hoped would happen by cheating, doesn't happen.

In the best of all possible worlds, every protocol would be self-enforcing. Unfortunately, there is not a self-enforcing protocol for every situation.

Attacks against Protocols

Cryptographic attacks can be directed against the cryptographic algorithms used in protocols, against the cryptographic techniques used to implement the algorithms and protocols, or against the protocols themselves. Since this section of the book discusses protocols, I will assume that the cryptographic algorithms and techniques are secure. I will only examine attacks against the protocols.

People can try various ways to attack a protocol. Someone not involved in the protocol can eavesdrop on some or all of the protocol. This is called a **passive attack**, because the attacker does not affect the protocol. All he can do is observe the protocol and attempt to gain information. This kind of attack corresponds to a ciphertext-only attack, as discussed in Section 1.1. Since passive attacks are difficult to detect, protocols try to prevent passive attacks rather than detect them. In these protocols, the part of the eavesdropper will be played by Eve.

Alternatively, an attacker could try to alter the protocol to his own advantage. He could pretend to be someone else, introduce new messages in the protocol, delete existing messages, substitute one message for another, replay old messages, interrupt a communications channel, or alter stored information in a computer. These are called **active attacks**, because they require active intervention. The form of these attacks depends on the network.

Passive attackers try to gain information about the parties involved in the protocol. They collect messages passing among various parties and attempt to cryptanalyze them. Active attacks, on the other hand, can have much more diverse objectives. The attacker could be interested in obtaining information, degrading system performance, corrupting existing information, or gaining unauthorized access to resources.

Active attacks are much more serious, especially in protocols in which the different parties don't necessarily trust one another. The attacker does not have to be a complete outsider. He could be a legitimate system user. He could be the system administrator. There could even be many active attackers working together. Here, the part of the malicious active attacker will be played by Mallory.

It is also possible that the attacker could be one of the parties involved in the protocol. He may lie during the protocol or not follow the protocol at all. This type of attacker is called a **cheater**. **Passive cheaters** follow the protocol, but try to obtain more information than the protocol intends them to. **Active cheaters** disrupt the protocol in progress in an attempt to cheat.

It is very difficult to maintain a protocol's security if most of the parties involved are active cheaters, but sometimes it is possible for legitimate parties to detect that active cheating is going on. Certainly, protocols should be secure against passive cheating.

2.2 Communications Using Symmetric Cryptography

How do two parties communicate securely? They encrypt their communications, of course. The complete protocol is more complicated than that. Let's look at what must happen for Alice to send an encrypted message to Bob.

- (1) Alice and Bob agree on a cryptosystem.
- (2) Alice and Bob agree on a key.
- (3) Alice takes her plaintext message and encrypts it using the

encryption algorithm and the key. This creates a ciphertext message.

(4) Alice sends the ciphertext message to Bob.

(5) Bob decrypts the ciphertext message with the same algorithm and key and reads it.

What can Eve, sitting between Alice and Bob, learn from listening in on this protocol? If all she hears is the transmission in step (4), she must try to cryptanalyze the ciphertext. This passive attack is a ciphertext-only attack; we have algorithms that are resistant (as far as we know) to whatever computing power Eve could realistically bring to bear on the problem.

Eve isn't stupid, though. She also wants to listen in on steps (1) and (2). Then, she would know the algorithm and the key—just as well as Bob. When the message comes across the communications channel in step (4), all she has to do is decrypt it herself.

A good cryptosystem is one in which all the security is inherent in knowledge of the key and none is inherent in knowledge of the algorithm. This is why key management is so important in cryptography. With a symmetric algorithm, Alice and Bob can perform step (1) in public, but they must perform step (2) in secret. The key must remain secret before, during, and after the protocol—as long as the message must remain secret—otherwise the message will no longer be secure. (Public-key cryptography solves this problem another way, and will be discussed in Section 2.5.)

Mallory, an active attacker, could do a few other things. He could attempt to break the communications path in step (4), ensuring that Alice could not talk to Bob at all. Mallory could also intercept Alice's messages and substitute his own. If he knew the key (by intercepting the communication in step (2), or by breaking the cryptosystem), he could encrypt his own message and send it to Bob in place of the intercepted message. Bob would have no way of knowing that the message had not come from Alice. If Mallory didn't know the key, he could only create a replacement message that would decrypt to gibberish. Bob, thinking the message came from Alice, might conclude that either the network or Alice had some serious problems.

Previous	Table of Contents	Next
--------------------------	-----------------------------------	----------------------

[Products](#) | [Contact Us](#) | [About Us](#) | [Privacy](#) | [Ad Info](#) | [Home](#)

Use of this site is subject to certain [Terms & Conditions](#), Copyright © 1996-2000 EarthWeb Inc.

All rights reserved. Reproduction whole or in part in any form or medium without express written [permission](#) of EarthWeb is prohibited. Read EarthWeb's [privacy](#) statement.

 SEARCH
ITKNOWLEDGE

☒ Brief
 ☐ Full

[Advanced Search](#)

[Search Tips](#)

 BROWSE
BY TOPIC

To access the contents, click the chapter and section titles.

Applied Cryptography, Second Edition: Protocols, Algorithms, and Source Code in C (cloth)

(Publisher: John Wiley & Sons, Inc.)

Author(s): Bruce Schneier

ISBN: 0471128457

Publication Date: 01/01/96

Bookmark It

Search this book:

[Previous](#)
[Table of Contents](#)
[Next](#)

What about Alice? What can she do to disrupt the protocol? She can give a copy of the key to Eve. Now Eve can read whatever Bob says. She can reprint his words in *The New York Times*. Although serious, this is not a problem with the protocol. There is nothing to stop Alice from giving Eve a copy of the plaintext at any point during the protocol. Of course, Bob could also do anything that Alice could. This protocol assumes that Alice and Bob trust each other.

In summary, symmetric cryptosystems have the following problems:

- Keys must be distributed in secret. They are as valuable as all the messages they encrypt, since knowledge of the key gives knowledge of all the messages. For encryption systems that span the world, this can be a daunting task. Often couriers hand-carry keys to their destinations.
- If a key is compromised (stolen, guessed, extorted, bribed, etc.), then Eve can decrypt all message traffic encrypted with that key. She can also pretend to be one of the parties and produce false messages to fool the other party.
- Assuming a separate key is used for each pair of users in a network, the total number of keys increases rapidly as the number of users increases. A network of n users requires $n(n - 1)/2$ keys. For example, 10 users require 45 different keys to talk with one another and 100 users require 4950 keys. This problem can be minimized by keeping the number of users small, but that is not always possible.

2.3 One-Way Functions

The notion of a **one-way function** is central to public-key cryptography. While not protocols in themselves, one-way functions are a fundamental building block for most of the protocols discussed in this book.

One-way functions are relatively easy to compute, but significantly harder to reverse. That is, given x it is easy to compute $f(x)$, but given $f(x)$ it is hard to compute x . In this context, “hard” is defined as something like: It would take millions of years to compute x from $f(x)$, even if all the computers in the world were assigned to the problem.

Breaking a plate is a good example of a one-way function. It is easy to smash a plate into a thousand tiny pieces. However, it’s not easy to put all of those tiny pieces back together into a plate.

This sounds good, but it’s a lot of smoke and mirrors. If we are being strictly mathematical, we have no proof that one-way functions exist, nor any real evidence that they can be constructed [230,530,600,661]. Even so, many functions look and smell one-way: We can compute them efficiently and, as of yet, know of no easy way to reverse them. For example, in a finite field x^2 is easy to compute, but $x^{1/2}$ is much harder. For the rest of this section, I’m going to pretend that there are one-way functions. I’ll talk more about this in Section 11.2.

So, what good are one-way functions? We can’t use them for encryption as is. A message encrypted with the one-way function isn’t useful; no one could decrypt it. (Exercise: Write a message on a plate, smash the plate into tiny bits, and then give the bits to a friend. Ask your friend to read the message. Observe how impressed he is with the one-way function.) For public-key cryptography, we need something else (although there are cryptographic applications for one-way functions—see Section 3.2).

A **trapdoor one-way function** is a special type of one-way function, one with a secret trapdoor. It is easy to compute in one direction and hard to compute in the other direction. But, if you know the secret, you can easily compute the function in the other direction. That is, it is easy to compute $f(x)$ given x , and hard to compute x given $f(x)$. However, there is some secret information, y , such that given $f(x)$ and y it is easy to compute x .

Taking a watch apart is a good example of a trap-door one-way function. It is easy to disassemble a watch into hundreds of minuscule pieces. It is very difficult to put those tiny pieces back together into a working watch. However, with the secret information—the assembly instructions of the watch—it is much easier to put the watch back together.

2.4 One-Way Hash Functions

A **one-way hash function** has many names: compression function, contraction function, message digest, fingerprint, cryptographic checksum, message integrity check (MIC), and manipulation detection code (MDC). Whatever you call it, it is central to modern cryptography. One-way hash functions are another building block for many protocols.

Hash functions have been used in computer science for a long time. A hash function is a function, mathematical or otherwise, that takes a variable-length input string (called a **pre-image**) and converts it to a fixed-length (generally smaller) output string (called a **hash value**). A simple hash function would be

a function that takes pre-image and returns a byte consisting of the XOR of all the input bytes.

The point here is to fingerprint the pre-image: to produce a value that indicates whether a candidate pre-image is likely to be the same as the real pre-image. Because hash functions are typically many-to-one, we cannot use them to determine with certainty that the two strings are equal, but we can use them to get a reasonable assurance of accuracy.

A one-way hash function is a hash function that works in one direction: It is easy to compute a hash value from pre-image, but it is hard to generate a pre-image that hashes to a particular value. The hash function previously mentioned is not one-way: Given a particular byte value, it is trivial to generate a string of bytes whose XOR is that value. You can't do that with a one-way hash function. A good one-way hash function is also **collision-free**: It is hard to generate two pre-images with the same hash value.

The hash function is public; there's no secrecy to the process. The security of a one-way hash function is its one-wayness. The output is not dependent on the input in any discernible way. A single bit change in the pre-image changes, on the average, half of the bits in the hash value. Given a hash value, it is computationally unfeasible to find a pre-image that hashes to that value.

Think of it as a way of fingerprinting files. If you want to verify that someone has a particular file (that you also have), but you don't want him to send it to you, then ask him for the hash value. If he sends you the correct hash value, then it is almost certain that he has that file. This is particularly useful in financial transactions, where you don't want a withdrawal of \$100 to turn into a withdrawal of \$1000 somewhere in the network. Normally, you would use a one-way hash function without a key, so that anyone can verify the hash. If you want only the recipient to be able to verify the hash, then read the next section.

Previous	Table of Contents	Next
--------------------------	-----------------------------------	----------------------

[Products](#) | [Contact Us](#) | [About Us](#) | [Privacy](#) | [Ad Info](#) | [Home](#)

Use of this site is subject to certain [Terms & Conditions](#). Copyright © 1996-2000 EarthWeb Inc.

All rights reserved. Reproduction whole or in part in any form or medium without express written [permission](#) of EarthWeb is prohibited. Read EarthWeb's [privacy](#) statement.

 SEARCH
ITKNOWLEDGE

☒ Brief
 ☐ Full

[Advanced Search](#)

[Search Tips](#)

 BROWSE
BY TOPIC

To access the contents, click the chapter and section titles.

Applied Cryptography, Second Edition: Protocols, Algorithms, and Source Code in C (cloth)

(Publisher: John Wiley & Sons, Inc.)

Author(s): Bruce Schneier

ISBN: 0471128457

Publication Date: 01/01/96

Bookmark It

Search this book:

[Previous](#)
[Table of Contents](#)
[Next](#)

Message Authentication Codes

A **message authentication code** (MAC), also known as a data authentication code (DAC), is a one-way hash function with the addition of a secret key (see Section 18.14). The hash value is a function of both the pre-image and the key. The theory is exactly the same as hash functions, except only someone with the key can verify the hash value. You can create a MAC out of a hash function or a block encryption algorithm; there are also dedicated MACs.

2.5 Communications Using Public-Key Cryptography

Think of a symmetric algorithm as a safe. The key is the combination. Someone with the combination can open the safe, put a document inside, and close it again. Someone else with the combination can open the safe and take the document out. Anyone without the combination is forced to learn safecracking.

In 1976, Whitfield Diffie and Martin Hellman changed that paradigm of cryptography forever [496]. (The NSA has claimed knowledge of the concept as early as 1966, but has offered no proof.) They described **public-key cryptography**. They used two different keys—one public and the other private. It is computationally hard to deduce the private key from the public key. Anyone with the public key can encrypt a message but not decrypt it. Only the person with the private key can decrypt the message. It is as if someone turned the cryptographic safe into a mailbox. Putting mail in the mailbox is analogous to encrypting with the public key; anyone can do it. Just open the slot and drop it in. Getting mail out of a mailbox is analogous to decrypting with the private key. Generally it's hard; you need welding torches. However, if you have the secret (the physical key to the mailbox), it's easy to

get mail out of a mailbox.

Mathematically, the process is based on the trap-door one-way functions previously discussed. Encryption is the easy direction. Instructions for encryption are the public key; anyone can encrypt a message. Decryption is the hard direction. It's made hard enough that people with Cray computers and thousands (even millions) of years couldn't decrypt the message without the secret. The secret, or trapdoor, is the private key. With that secret, decryption is as easy as encryption.

This is how Alice can send a message to Bob using public-key cryptography:

- (1) Alice and Bob agree on a public-key cryptosystem.
- (2) Bob sends Alice his public key.
- (3) Alice encrypts her message using Bob's public key and sends it to Bob.
- (4) Bob decrypts Alice's message using his private key.

Notice how public-key cryptography solves the key-management problem with symmetric cryptosystems. Before, Alice and Bob had to agree on a key in secret. Alice could choose one at random, but she still had to get it to Bob. She could hand it to him sometime beforehand, but that requires foresight. She could send it to him by secure courier, but that takes time. Public-key cryptography makes it easy. With no prior arrangements, Alice can send a secure message to Bob. Eve, listening in on the entire exchange, has Bob's public key and a message encrypted in that key, but cannot recover either Bob's private key or the message.

More commonly, a network of users agrees on a public-key cryptosystem. Every user has his or her own public key and private key, and the public keys are all published in a database somewhere. Now the protocol is even easier:

- (1) Alice gets Bob's public key from the database.
- (2) Alice encrypts her message using Bob's public key and sends it to Bob.
- (3) Bob then decrypts Alice's message using his private key.

In the first protocol, Bob had to send Alice his public key before she could send him a message. The second protocol is more like traditional mail. Bob is not involved in the protocol until he wants to read his message.

Hybrid Cryptosystems

The first public-key algorithms became public at the same time that DES was being discussed as a proposed standard. This resulted in some partisan politics in the cryptographic community. As Diffie described it [494]:

The excitement public key cryptosystems provoked in the popular and scientific press was not matched by corresponding acceptance in the cryptographic establishment, however. In the same year that public key cryptography was discovered, the National Security Agency (NSA), proposed a conventional cryptographic system, designed by International Business Machines (IBM), as a

federal *Data Encryption Standard* (DES). Marty Hellman and I criticized the proposal on the ground that its key was too small, but manufacturers were gearing up to support the proposed standard and our criticism was seen by many as an attempt to disrupt the standards-making process to the advantage of our own work. Public key cryptography in its turn was attacked, in sales literature [1125] and technical papers [849,1159] alike, more as though it were a competing product than a recent research discovery. This, however, did not deter the NSA from claiming its share of the credit. Its director, in the words of the *Encyclopedia Britannica* [1461], pointed out that “two-key cryptography had been discovered at the agency a decade earlier,” although no evidence for this claim was ever offered publicly.

In the real world, public-key algorithms are not a substitute for symmetric algorithms. They are not used to encrypt messages; they are used to encrypt keys. There are two reasons for this:

1. Public-key algorithms are slow. Symmetric algorithms are generally at least 1000 times faster than public-key algorithms. Yes, computers are getting faster and faster, and in 15 years computers will be able to do public-key cryptography at speeds comparable to symmetric cryptography today. But bandwidth requirements are also increasing, and there will always be the need to encrypt data faster than public-key cryptography can manage.
2. Public-key cryptosystems are vulnerable to chosen-plaintext attacks. If $C = E(P)$, when P is one plaintext out of a set of n possible plaintexts, then a cryptanalyst only has to encrypt all n possible plaintexts and compare the results with C (remember, the encryption key is public). He won't be able to recover the decryption key this way, but he will be able to determine P .

Previous	Table of Contents	Next
--------------------------	-----------------------------------	----------------------

[Products](#) | [Contact Us](#) | [About Us](#) | [Privacy](#) | [Ad Info](#) | [Home](#)

Use of this site is subject to certain [Terms & Conditions](#). Copyright © 1996-2000 EarthWeb Inc.

All rights reserved. Reproduction whole or in part in any form or medium without express written [permission](#) of EarthWeb is prohibited. Read EarthWeb's [privacy](#) statement.

 SEARCH
ITKNOWLEDGE

☒ Brief
 ☐ Full

[Advanced Search](#)

[Search Tips](#)

 BROWSE
BY TOPIC

To access the contents, click the chapter and section titles.

Applied Cryptography, Second Edition: Protocols, Algorithms, and Source Code in C (cloth)

(Publisher: John Wiley & Sons, Inc.)

Author(s): Bruce Schneier

ISBN: 0471128457

Publication Date: 01/01/96

Bookmark It

Search this book:

[Previous](#)
[Table of Contents](#)
[Next](#)

A chosen-plaintext attack can be particularly effective if there are relatively few possible encrypted messages. For example, if P were a dollar amount less than \$1,000,000, this attack would work; the cryptanalyst tries all million possible dollar amounts. (Probabilistic encryption solves the problem; see Section 23.15.) Even if P is not as well-defined, this attack can be very effective. Simply knowing that a ciphertext does not correspond to a particular plaintext can be useful information. Symmetric cryptosystems are not vulnerable to this attack because a cryptanalyst cannot perform trial encryptions with an unknown key.

In most practical implementations public-key cryptography is used to secure and distribute **session keys**; those session keys are used with symmetric algorithms to secure message traffic [879]. This is sometimes called a **hybrid cryptosystem**.

- (1) Bob sends Alice his public key.
- (2) Alice generates a random session key, K , encrypts it using Bob's public key, and sends it to Bob.

$$E_B(K)$$

- (3) Bob decrypts Alice's message using his private key to recover the session key.

$$D_B(E_B(K)) = K$$

- (4) Both of them encrypt their communications using the same session key.

Using public-key cryptography for key distribution solves a very important key-management problem. With symmetric cryptography, the data encryption key sits around until it is used. If Eve ever gets her hands on it, she can decrypt messages encrypted with it. With the previous protocol, the session key is created when it is needed to encrypt communications and destroyed when it is

no longer needed. This drastically reduces the risk of compromising the session key. Of course, the private key is vulnerable to compromise, but it is at less risk because it is only used once per communication to encrypt a session key. This is further discussed in Section 3.1.

Merkle's Puzzles

Ralph Merkle invented the first construction of public-key cryptography. In 1974 he registered for a course in computer security at the University of California, Berkeley, taught by Lance Hoffman. His term paper topic, submitted early in the term, addressed the problem of “Secure Communication over Insecure Channels” [1064]. Hoffman could not understand Merkle’s proposal and eventually Merkle dropped the course. He continued to work on the problem, despite continuing failure to make his results understood.

Merkle’s technique was based on “puzzles” that were easier to solve for the sender and receiver than for an eavesdropper. Here’s how Alice sends an encrypted message to Bob without first having to exchange a key with him.

- (1) Bob generates 2^{20} , or about a million, messages of the form: “This is puzzle number x . This is the secret key number y ,” where x is a random number and y is a random secret key. Both x and y are different for each message. Using a symmetric algorithm, he encrypts each message with a different 20-bit key and sends them all to Alice.
- (2) Alice chooses one message at random and performs a brute-force attack to recover the plaintext. This is a large, but not impossible, amount of work.
- (3) Alice encrypts her secret message with the key she recovered and some symmetric algorithm, and sends it to Bob along with x .
- (4) Bob knows which secret key y he encrypts in message x , so he can decrypt the message.

Eve can break this system, but she has to do far more work than either Alice or Bob. To recover the message in step (3), she has to perform a brute-force attack against each of Bob’s 2^{20} messages in step (1); this attack has a complexity of 2^{40} . The x values won’t help Eve either; they were assigned randomly in step (1). In general, Eve has to expend approximately the square of the effort that Alice expends.

This n to n^2 advantage is small by cryptographic standards, but in some circumstances it may be enough. If Alice and Bob can try ten thousand keys per second, it will take them a minute each to perform their steps and another minute to communicate the puzzles from Bob to Alice on a 1.544 MB link. If Eve had comparable computing facilities, it would take her about a year to break the system. Other algorithms are even harder to break.

2.6 Digital Signatures

Handwritten signatures have long been used as proof of authorship of, or at least agreement with, the contents of a document. What is it about a signature that is so compelling [1392]?

1. The signature is authentic. The signature convinces the document's recipient that the signer deliberately signed the document.
2. The signature is unforgeable. The signature is proof that the signer, and no one else, deliberately signed the document.
3. The signature is not reusable. The signature is part of the document; an unscrupulous person cannot move the signature to a different document.
4. The signed document is unalterable. After the document is signed, it cannot be altered.
5. The signature cannot be repudiated. The signature and the document are physical things. The signer cannot later claim that he or she didn't sign it.

In reality, none of these statements about signatures is completely true. Signatures can be forged, signatures can be lifted from one piece of paper and moved to another, and documents can be altered after signing. However, we are willing to live with these problems because of the difficulty in cheating and the risk of detection.

We would like to do this sort of thing on computers, but there are problems. First, computer files are trivial to copy. Even if a person's signature were difficult to forge (a graphical image of a written signature, for example), it would be easy to cut and paste a valid signature from one document to another document. The mere presence of such a signature means nothing. Second, computer files are easy to modify after they are signed, without leaving any evidence of modification.

Previous	Table of Contents	Next
--------------------------	-----------------------------------	----------------------

[Products](#) | [Contact Us](#) | [About Us](#) | [Privacy](#) | [Ad Info](#) | [Home](#)

Use of this site is subject to certain [Terms & Conditions](#), [Copyright © 1996-2000 EarthWeb Inc.](#)

All rights reserved. Reproduction whole or in part in any form or medium without express written [permission](#) of EarthWeb is prohibited. Read EarthWeb's [privacy](#) statement.

☒ Brief
 ☐ Full

[Advanced Search](#)

[Search Tips](#)

 **BROWSE**
BY TOPIC

To access the contents, click the chapter and section titles.

Applied Cryptography, Second Edition: Protocols, Algorithms, and Source Code in C (cloth)

(Publisher: John Wiley & Sons, Inc.)

Author(s): Bruce Schneier

ISBN: 0471128457

Publication Date: 01/01/96

Bookmark It

Search this book:

[Previous](#)
[Table of Contents](#)
[Next](#)

Signing Documents with Symmetric Cryptosystems and an Arbitrator

Alice wants to sign a digital message and send it to Bob. With the help of Trent and a symmetric cryptosystem, she can.

Trent is a powerful, trusted arbitrator. He can communicate with both Alice and Bob (and everyone else who may want to sign a digital document). He shares a secret key, K_A , with Alice, and a different secret key, K_B , with Bob. These keys have been established long before the protocol begins and can be reused multiple times for multiple signings.

- (1) Alice encrypts her message to Bob with K_A and sends it to Trent.
- (2) Trent decrypts the message with K_A .
- (3) Trent takes the decrypted message and a statement that he has received this message from Alice, and encrypts the whole bundle with K_B .
- (4) Trent sends the encrypted bundle to Bob.
- (5) Bob decrypts the bundle with K_B . He can now read both the message and Trent's certification that Alice sent it.

How does Trent know that the message is from Alice and not from some imposter? He infers it from the message's encryption. Since only he and Alice share their secret key, only Alice could encrypt a message using it.

Is this as good as a paper signature? Let's look at the characteristics we want:

1. This signature is authentic. Trent is a trusted arbitrator and Trent knows that the message came from Alice. Trent's certification serves as proof to Bob.

2. This signature is unforgeable. Only Alice (and Trent, but everyone trusts him) knows K_A , so only Alice could have sent Trent a message encrypted with K_A . If someone tried to impersonate Alice, Trent would have immediately realized this in step (2) and would not certify its authenticity.
3. This signature is not reusable. If Bob tried to take Trent's certification and attach it to another message, Alice would cry foul. An arbitrator (it could be Trent or it could be a completely different arbitrator with access to the same information) would ask Bob to produce both the message and Alice's encrypted message. The arbitrator would then encrypt the message with K_A and see that it did not match the encrypted message that Bob gave him. Bob, of course, could not produce an encrypted message that matches because he does not know K_A .
4. The signed document is unalterable. Were Bob to try to alter the document after receipt, Trent could prove foul play in exactly the same manner just described.
5. The signature cannot be repudiated. Even if Alice later claims that she never sent the message, Trent's certification says otherwise. Remember, Trent is trusted by everyone; what he says is true.

If Bob wants to show Carol a document signed by Alice, he can't reveal his secret key to her. He has to go through Trent again:

- (1) Bob takes the message and Trent's statement that the message came from Alice, encrypts them with K_B , and sends them back to Trent.
- (2) Trent decrypts the bundle with K_B .
- (3) Trent checks his database and confirms that the original message came from Alice.
- (4) Trent re-encrypts the bundle with the secret key he shares with Carol, K_C , and sends it to Carol.
- (5) Carol decrypts the bundle with K_C . She can now read both the message and Trent's certification that Alice sent it.

These protocols work, but they're time-consuming for Trent. He must spend his days decrypting and encrypting messages, acting as the intermediary between every pair of people who want to send signed documents to one another. He must keep a database of messages (although this can be avoided by sending the recipient a copy of the sender's encrypted message). He is a bottleneck in any communications system, even if he's a mindless software program.

Harder still is creating and maintaining someone like Trent, someone that everyone on the network trusts. Trent has to be infallible; if he makes even one mistake in a million signatures, no one is going to trust him. Trent has to be completely secure. If his database of secret keys ever got out or if someone managed to modify his programming, everyone's signatures would be completely useless. False documents purported to be signed years ago could appear. Chaos would result. Governments would collapse. Anarchy would

reign. This might work in theory, but it doesn't work very well in practice.

Digital Signature Trees

Ralph Merkle proposed a digital signature scheme based on secret-key cryptography, producing an infinite number of one-time signatures using a tree structure [1067,1068]. The basic idea of this scheme is to place the root of the tree in some public file, thereby authenticating it. The root signs one message and authenticates its sub-nodes in the tree. Each of these nodes signs one message and authenticates its sub-nodes, and so on.

Signing Documents with Public-Key Cryptography

There are public-key algorithms that can be used for digital signatures. In some algorithms—RSA is an example (see Section 19.3)—either the public key or the private key can be used for encryption. Encrypt a document using your private key, and you have a secure digital signature. In other cases—DSA is an example (see Section 20.1)—there is a separate algorithm for digital signatures that cannot be used for encryption. This idea was first invented by Diffie and Hellman [496] and further expanded and elaborated on in other texts [1282,1328,1024,1283,426]. See [1099] for a good survey of the field.

The basic protocol is simple:

- (1) Alice encrypts the document with her private key, thereby signing the document.
- (2) Alice sends the signed document to Bob.
- (3) Bob decrypts the document with Alice's public key, thereby verifying the signature.

This protocol is far better than the previous one. Trent is not needed to either sign or verify signatures. (He is needed to certify that Alice's public key is indeed her public key.) The parties do not even need Trent to resolve disputes: If Bob cannot perform step (3), then he knows the signature is not valid.

Previous	Table of Contents	Next
--------------------------	-----------------------------------	----------------------

[Products](#) | [Contact Us](#) | [About Us](#) | [Privacy](#) | [Ad Info](#) | [Home](#)

Use of this site is subject to certain [Terms & Conditions](#), [Copyright © 1996-2000 EarthWeb Inc.](#)

All rights reserved. Reproduction whole or in part in any form or medium without express written [permission](#) of EarthWeb is prohibited. Read EarthWeb's [privacy](#) statement.

SEARCH
ITKNOWLEDGE

Keyword

☒ Brief ☐ Full

[Advanced Search](#)

[Search Tips](#)

BROWSE
BY TOPIC

To access the contents, click the chapter and section titles.

Applied Cryptography, Second Edition: Protocols, Algorithms, and Source Code in C (cloth)

(Publisher: John Wiley & Sons, Inc.)

Author(s): Bruce Schneier

ISBN: 0471128457

Publication Date: 01/01/96

[Bookmark It](#)

Search this book:

[Previous](#) [Table of Contents](#) [Next](#)

This protocol also satisfies the characteristics we're looking for:

1. The signature is authentic; when Bob verifies the message with Alice's public key, he knows that she signed it.
2. The signature is unforgeable; only Alice knows her private key.
3. The signature is not reusable; the signature is a function of the document and cannot be transferred to any other document.
4. The signed document is unalterable; if there is any alteration to the document, the signature can no longer be verified with Alice's public key.
5. The signature cannot be repudiated. Bob doesn't need Alice's help to verify her signature.

Signing Documents and Timestamps

Actually, Bob can cheat Alice in certain circumstances. He can reuse the document and signature together. This is no problem if Alice signed a contract (what's another copy of the same contract, more or less?), but it can be very exciting if Alice signed a digital check.

Let's say Alice sends Bob a signed digital check for \$100. Bob takes the check to the bank, which verifies the signature and moves the money from one account to the other. Bob, who is an unscrupulous character, saves a copy of the digital check. The following week, he again takes it to the bank (or maybe to a different bank). The bank verifies the signature and moves the money from one account to the other. If Alice never balances her checkbook, Bob can keep this up for years.

Consequently, digital signatures often include timestamps. The date and time of the signature are attached to the message and signed along with the rest of the message. The bank stores this timestamp in a database. Now, when Bob

tries to cash Alice's check a second time, the bank checks the timestamp against its database. Since the bank already cashed a check from Alice with the same timestamp, the bank calls the police. Bob then spends 15 years in Leavenworth prison reading up on cryptographic protocols.

Signing Documents with Public-Key Cryptography and One-Way Hash Functions

In practical implementations, public-key algorithms are often too inefficient to sign long documents. To save time, digital signature protocols are often implemented with one-way hash functions [432,433]. Instead of signing a document, Alice signs the hash of the document. In this protocol, both the one-way hash function and the digital signature algorithm are agreed upon beforehand.

- (1) Alice produces a one-way hash of a document.
- (2) Alice encrypts the hash with her private key, thereby signing the document.
- (3) Alice sends the document and the signed hash to Bob.
- (4) Bob produces a one-way hash of the document that Alice sent. He then, using the digital signature algorithm, decrypts the signed hash with Alice's public key. If the signed hash matches the hash he generated, the signature is valid.

Speed increases drastically and, since the chances of two different documents having the same 160-bit hash are only one in 2^{160} , anyone can safely equate a signature of the hash with a signature of the document. If a non-one-way hash function were used, it would be an easy matter to create multiple documents that hashed to the same value, so that anyone signing a particular document would be duped into signing a multitude of documents.

This protocol has other benefits. First, the signature can be kept separate from the document. Second, the recipient's storage requirements for the document and signature are much smaller. An archival system can use this type of protocol to verify the existence of documents without storing their contents. The central database could just store the hashes of files. It doesn't have to see the files at all; users submit their hashes to the database, and the database timestamps the submissions and stores them. If there is any disagreement in the future about who created a document and when, the database could resolve it by finding the hash in its files. This system has vast implications concerning privacy: Alice could copyright a document but still keep the document secret. Only if she wished to prove her copyright would she have to make the document public. (See Section 4.1).

Algorithms and Terminology

There are many digital signature algorithms. All of them are public-key algorithms with secret information to sign documents and public information to verify signatures. Sometimes the signing process is called **encrypting with a private key** and the verification process is called **decrypting with a public key**. This is misleading and is only true for one algorithm, RSA. And different

algorithms have different implementations. For example, one-way hash functions and timestamps sometimes add extra steps to the process of signing and verifying. Many algorithms can be used for digital signatures, but not for encryption.

In general, I will refer to the signing and verifying processes without any details of the algorithms involved. Signing a message with private key K is:

$$S_K(M)$$

and verifying a signature with the corresponding public key is:

$$V_K(M)$$

The bit string attached to the document when signed (in the previous example, the one-way hash of the document encrypted with the private key) will be called the **digital signature**, or just the **signature**. The entire protocol, by which the receiver of a message is convinced of the identity of the sender and the integrity of the message, is called authentication. Further details on these protocols are in Section 3.2.

Multiple Signatures

How could Alice and Bob sign the same digital document? Without one-way hash functions, there are two options. One is that Alice and Bob sign separate copies of the document itself. The resultant message would be over twice the size of the original document. The second is that Alice signs the document first and then Bob signs Alice's signature. This works, but it is impossible to verify Alice's signature without also verifying Bob's.

With one-way hash functions, multiple signatures are easy:

- (1) Alice signs the hash of the document.
- (2) Bob signs the hash of the document.
- (3) Bob sends his signature to Alice.
- (4) Alice sends the document, her signature, and Bob's signature to Carol.
- (5) Carol verifies both Alice's signature and Bob's signature.

Alice and Bob can do steps (1) and (2) either in parallel or in series. In step (5), Carol can verify one signature without having to verify the other.

Previous	Table of Contents	Next
--------------------------	-----------------------------------	----------------------

[Products](#) | [Contact Us](#) | [About Us](#) | [Privacy](#) | [Ad Info](#) | [Home](#)

Use of this site is subject to certain [Terms & Conditions](#), [Copyright](#) © 1996-2000 EarthWeb Inc.

All rights reserved. Reproduction whole or in part in any form or medium without express written [permission](#) of EarthWeb is prohibited. Read EarthWeb's [privacy](#) statement.

☒ Brief
 ☐ Full

[Advanced Search](#)

[Search Tips](#)

 **BROWSE**
BY TOPIC

To access the contents, click the chapter and section titles.

Applied Cryptography, Second Edition: Protocols, Algorithms, and Source Code in C (cloth)

(Publisher: John Wiley & Sons, Inc.)

Author(s): Bruce Schneier

ISBN: 0471128457

Publication Date: 01/01/96

Bookmark It

Search this book:

[Previous](#)
[Table of Contents](#)
[Next](#)

Nonrepudiation and Digital Signatures

Alice can cheat with digital signatures and there's nothing that can be done about it. She can sign a document and then later claim that she did not. First, she signs the document normally. Then, she anonymously publishes her private key, conveniently loses it in a public place, or just pretends to do either one. Alice then claims that her signature has been compromised and that others are using it, pretending to be her. She disavows signing the document and any others that she signed using that private key. This is called repudiation.

Timestamps can limit the effects of this kind of cheating, but Alice can always claim that her key was compromised earlier. If Alice times things well, she can sign a document and then successfully claim that she didn't. This is why there is so much talk about private keys buried in tamper-resistant modules—so that Alice can't get at hers and abuse it.

Although nothing can be done about this possible abuse, one can take steps to guarantee that old signatures are not invalidated by actions taken in disputing new ones. (For example, Alice could "lose" her key to keep from paying Bob for the junk car he sold her yesterday and, in the process, invalidate her bank account.) The solution is for the receiver of a signed document to have it timestamped [453].

The general protocol is given in [28]:

- (1) Alice signs a message.
- (2) Alice generates a header containing some identifying information. She concatenates the header with the signed message, signs that, and sends it to Trent.
- (3) Trent verifies the outside signature and confirms the identifying information. He adds a timestamp to Alice's signed message and the

identifying information. Then he signs it all and sends it to both Alice and Bob.

(4) Bob verifies Trent's signature, the identifying information, and Alice's signature.

(5) Alice verifies the message Trent sent to Bob. If she did not originate the message, she speaks up quickly.

Another scheme uses Trent after the fact [209]. After receiving a signed message, Bob can send a copy to Trent for verification. Trent can attest to the validity of Alice's signature.

Applications of Digital Signatures

One of the earliest proposed applications of digital signatures was to facilitate the verification of nuclear test ban treaties [1454,1467]. The United States and the Soviet Union (anyone remember the Soviet Union?) permitted each other to put seismometers on the other's soil to monitor nuclear tests. The problem was that each country needed to assure itself that the host nation was not tampering with the data from the monitoring nation's seismometers. Simultaneously, the host nation needed to assure itself that the monitor was sending only the specific information needed for monitoring.

Conventional authentication techniques can solve the first problem, but only digital signatures can solve both problems. The host nation can read, but not alter, data from the seismometer, and the monitoring nation knows that the data has not been tampered with.

2.7 Digital Signatures with Encryption

By combining digital signatures with public-key cryptography, we develop a protocol that combines the security of encryption with the authenticity of digital signatures. Think of a letter from your mother: The signature provides proof of authorship and the envelope provides privacy.

(1) Alice signs the message with her private key.

$$S_A(M)$$

(2) Alice encrypts the signed message with Bob's public key and sends it to Bob.

$$E_B(S_A(M))$$

(3) Bob decrypts the message with his private key.

$$D_B(E_B(S_A(M))) = S_A(M)$$

(4) Bob verifies with Alice's public key and recovers the message.

$$V_A(S_A(M)) = M$$

Signing before encrypting seems natural. When Alice writes a letter, she signs it and then puts it in an envelope. If she put the letter in the envelope unsigned and then signed the envelope, then Bob might worry if the letter hadn't been covertly replaced. If Bob showed to Carol Alice's letter and envelope, Carol might accuse Bob of lying about which letter arrived in which envelope.

In electronic correspondence as well, signing before encrypting is a prudent practice [48]. Not only is it more secure—an adversary can't remove a signature from an encrypted message and add his own—but there are legal considerations: If the text to be signed is not visible to the signer when he affixes his signature, then the signature may have little legal force [1312]. And there are some cryptanalytic attacks against this technique with RSA signatures (see Section 19.3).

There's no reason Alice has to use the same public-key/private-key key pair for encrypting and signing. She can have two key pairs: one for encryption and the other for signatures. Separation has its advantages: she can surrender her encryption key to the police without compromising her signature, one key can be escrowed (see Section 4.13) without affecting the other, and the keys can have different sizes and can expire at different times.

Of course, timestamps should be used with this protocol to prevent reuse of messages. Timestamps can also protect against other potential pitfalls, such as the one described below.

Previous	Table of Contents	Next
--------------------------	-----------------------------------	----------------------

[Products](#) | [Contact Us](#) | [About Us](#) | [Privacy](#) | [Ad Info](#) | [Home](#)

Use of this site is subject to certain [Terms & Conditions](#), [Copyright © 1996-2000 EarthWeb Inc.](#)

All rights reserved. Reproduction whole or in part in any form or medium without express written [permission](#) of EarthWeb is prohibited. Read EarthWeb's [privacy](#) statement.

devices.

— France has no special rules for the import of cryptography, but they have rules regarding the sale and use of cryptography in their country. All products must be certified: Either they must meet a published specification, or the company proprietary specification must be provided to the government. The government may also ask for two units for their own use. Companies must have a license to sell cryptography within France; the license specifies the target market. Users must have a license to buy and use cryptography; the license includes a statement to the effect that users must be prepared to give up their keys to the government up to four months after use. This restriction may be waived in some cases: for banks, large companies, and so on. And there is no use license requirement for cryptography exportable from the U.S.

— Germany follows the CoCom guidelines, requiring a license to export cryptography. They specifically maintain control of public-domain and mass-market cryptography software.

— Israel has import restrictions, but no one seems to know what they are.

— Belgium, Italy, Japan, Netherlands, and the United Kingdom follow the CoCom guidelines on cryptography, requiring a license for export.

— Brazil, India, Mexico, Russia, Saudi Arabia, Spain, South Africa, Sweden, and Switzerland have no import or export controls on cryptography.

25.16 Legal Issues

Are digital signatures real signatures? Will they stand up in court? Some preliminary legal research has resulted in the opinion that digital signatures would meet the requirements of legally binding signatures for most purposes, including commercial use as defined in the Uniform Commercial Code (UCC). A GAO (General Accounting Office) decision, made at the request of NIST, opines that digital signatures will meet the legal standards of handwritten signatures [362].

The Utah Digital Signature Act went into effect on May 1, 1995, providing a legal framework for the use of digital signatures in the judicial system. California has a bill pending, while Oregon and Washington are still writing theirs. Texas and Florida are right behind. By this book's publication, more states will have followed suit.

The American Bar Association (EDI and Information Technology Division of the Science and Technology Section) produced a model act for states to use for their own legislation. The act attempts to incorporate digital signatures into the existing legal infrastructure for signatures: the Uniform Commercial Code, the United States Federal Reserve regulations, common law of contracts and signatures, the United Nations Convention on Contracts for the International Sale of Goods, and the United Nations Convention on International Bills of Exchange and International Promissory Committees. Included in the act are responsibilities and obligations of certification authorities, issues of liability, and limits and policies.

In the United States, laws about signatures, contracts, and commercial transactions are state laws, so this model act is designed for states. The eventual goal is a federal act, but if this all begins at the state level there is less chance of the NSA mucking up the works.

Even so, the validity of digital signatures has not been challenged in court; their legal status is still undefined. In order for digital signatures to carry the same authority as handwritten signatures, they must first be used to sign a legally binding document, and then be challenged in court by one party. The court would then consider the security of the signature scheme and issue a ruling. Over time, as this happened repeatedly, a body of precedent rulings would emerge regarding which digital signature methods and what key sizes are required for a digital signature to be legally binding. This is likely to take years.

Until then, if two people wish to use digital signatures for contracts (or purchase requests, or work orders, or whatever), it is recommended that they sign a paper contract in which they agree in the future to be bound by any documents digitally signed by them [1099]. This document would specify algorithm, key size, and any other parameters; it should also delineate how disputes would be resolved.

Previous	Table of Contents	Next
--------------------------	-----------------------------------	----------------------

[Products](#) | [Contact Us](#) | [About Us](#) | [Privacy](#) | [Ad Info](#) | [Home](#)

Use of this site is subject to certain [Terms & Conditions](#), [Copyright © 1996-2000 EarthWeb Inc.](#)

All rights reserved. Reproduction whole or in part in any form or medium without express written [permission](#) of EarthWeb is prohibited. Read EarthWeb's [privacy](#) statement.