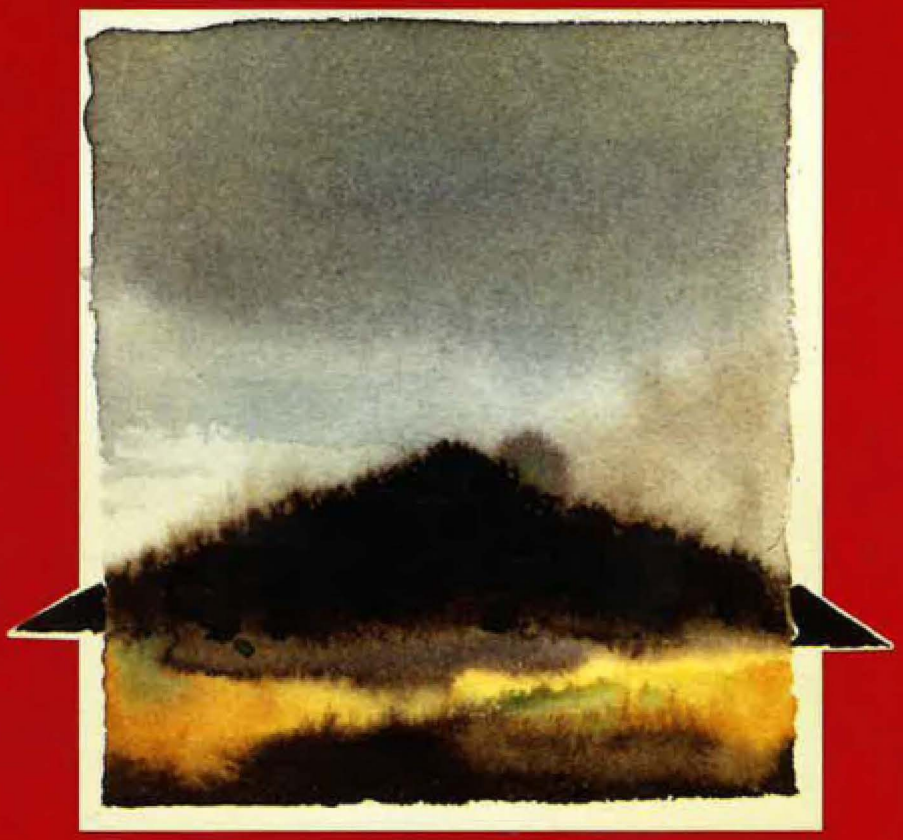
"... the best introduction to cryptography I've ever seen.... The book the National Security Agency wanted never to be published...."

—*Wired* magazine

MORE THAN 100,000 COPIES SOLD

SECOND EDITION

APPLIED CRYPTOGRAPHY



Protocols, Algorithms, and Source Code in C

BRUCE SCHNEIER

PLAINTIFF'S EXHIBIT
293

Election Systems & Software, LLC - Exhibit 1009
Election Systems & Software, LLC v. Hart Intercivic, Inc.

APPLIED CRYPTOGRAPHY, SECOND EDITION

PROTOCOLS, ALGORITHMS, AND SOURCE CODE IN C

BRUCE SCHNEIER



John Wiley & Sons, Inc.
New York • Chichester • Brisbane • Toronto • Singapore

Election Systems & Software, LLC - Exhibit 1009
Election Systems & Software, LLC v. Hart Intercivic, Inc.
Page 2

Publisher: Katherine Schowalter
Editor: Phil Sutherland
Assistant Editor: Allison Roarty
Managing Editor: Robert Aronds
Text Design & Composition: North Market Street Graphics

Designations used by companies to distinguish their products are often claimed as trademarks. In all instances where John Wiley & Sons, Inc. is aware of a claim, the product names appear in initial capital or all capital letters. Readers, however, should contact the appropriate companies for more complete information regarding trademarks and registration.

This text is printed on acid-free paper.

Copyright © 1996 by Bruce Schneier
Published by John Wiley & Sons, Inc.

All rights reserved. Published simultaneously in Canada.

This publication is designed to provide accurate and authoritative information in regard to the subject matter covered. It is sold with the understanding that the publisher is not engaged in rendering legal, accounting, or other professional service. If legal advice or other expert assistance is required, the services of a competent professional person should be sought.

In no event will the publisher or author be liable for any consequential, incidental, or indirect damages (including damages for loss of business profits, business interruption, loss of business information, and the like) arising from the use or inability to use the protocols and algorithms in this book, even if the publisher or author has been advised of the possibility of such damages.

Some of the protocols and algorithms in this book are protected by patents and copyrights. It is the responsibility of the reader to obtain all necessary patent and copyright licenses before implementing in software any protocol or algorithm in this book. This book does not contain an exhaustive list of all applicable patents and copyrights.

Some of the protocols and algorithms in this book are regulated under the United States Department of State International Traffic in Arms Regulations. It is the responsibility of the reader to obtain all necessary export licenses before implementing in software for export any protocol or algorithm in this book.

Reproduction or translation of any part of this work beyond that permitted by section 107 or 108 of the 1976 United States Copyright Act without the permission of the copyright owner is unlawful. Requests for permission or further information should be addressed to the Permissions Department, John Wiley & Sons, Inc.

Library of Congress Cataloging-in-Publication Data:

Schneier, Bruce
Applied Cryptography Second Edition : protocols, algorithms, and source code in C / Bruce Schneier.
p. cm.
Includes bibliographical references (p. 675).
ISBN 0-471-12845-7 (cloth : acid-free paper). — ISBN 0-471-11709-9 (paper : acid-free paper)
1. Computer security. 2. Telecommunication—Security measures. 3. Cryptography. I. Title.
QA76.9.A25S35 1996
005.8'2—dc20
95-12398
CIP

Printed in the United States of America
20 19 18

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.

Election Systems & Software, LLC - Exhibit 1009
Election Systems & Software, LLC v. Hart Intercivic, Inc.
Page 4

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.

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.

Election Systems & Software, LLC - Exhibit 1009
Election Systems & Software, LLC v. Hart Intercivic, Inc.
Page 5

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

Election Systems & Software, LLC - Exhibit 1009
Election Systems & Software, LLC v. Hart Intercivic, Inc.
Page 6

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.

Election Systems & Software, LLC - Exhibit 1009
Election Systems & Software, LLC v. Hart Intercivic, Inc.
Page 7

CHAPTER 6

Esoteric Protocols

6.1 SECURE ELECTIONS

Computerized voting will never be used for general elections unless there is a protocol that both maintains individual privacy and prevents cheating. The ideal protocol has, at the very least, these six requirements:

1. Only authorized voters can vote.
2. No one can vote more than once.
3. No one can determine for whom anyone else voted.
4. No one can duplicate anyone else's vote. (This turns out to be the hardest requirement.)
5. No one can change anyone else's vote without being discovered.
6. Every voter can make sure that his vote has been taken into account in the final tabulation.

Additionally, some voting schemes may have the following requirement:

7. Everyone knows who voted and who didn't.

Before describing the complicated voting protocols with these characteristics, let's look at some simpler protocols.

Simplistic Voting Protocol #1

(1) Each voter encrypts his vote with the public key of a Central Tabulating Facility (CTF).
(2) Each voter sends his vote in to the CTF.
(3) The CTF decrypts the votes, tabulates them, and makes the results public.

Election Systems & Software, LLC - Exhibit 1009
Election Systems & Software, LLC v. Hart Intercivic, Inc.
Page 8

This protocol is rife with problems. The CTF has no idea where the votes are from, so it doesn't even know if the votes are coming from eligible voters. It has no idea if eligible voters are voting more than once. On the plus side, no one can change anyone else's vote; but no one would bother trying to modify someone else's vote when it is far easier to vote repeatedly for the result of your choice.

Simplistic Voting Protocol #2

(1) Each voter signs his vote with his private key.

(2) Each voter encrypts his signed vote with the CTF's public key.

(3) Each voter sends his vote to a CTF.

(4) The CTF decrypts the votes, checks the signatures, tabulates the votes, and makes the results public.

This protocol satisfies properties one and two: Only authorized voters can vote and no one can vote more than once—the CTF would record votes received in step (3). Each vote is signed with the voter's private key, so the CTF knows who voted, who didn't, and how often each voter voted. If a vote comes in that isn't signed by an eligible voter, or if a second vote comes in signed by a voter who has already voted, the facility ignores it. No one can change anyone else's vote either, even if they intercept it in step (3), because of the digital signature.

The problem with this protocol is that the signature is attached to the vote; the CTF knows who voted for whom. Encrypting the votes with the CTF's public key prevents anyone from eavesdropping on the protocol and figuring out who voted for whom, but you have to trust the CTF completely. It's analogous to having an election judge staring over your shoulder in the voting booth.

These two examples show how difficult it is to achieve the first three requirements of a secure voting protocol, let alone the others.

Voting with Blind Signatures

We need to somehow dissociate the vote from the voter, while still maintaining authentication. The blind signature protocol does just that.

(1) Each voter generates 10 sets of messages, each set containing a valid vote for each possible outcome (e.g., if the vote is a yes or no question, each set contains two votes, one for "yes" and the other for "no"). Each message also contains a randomly generated identification number, large enough to avoid duplicates with other voters.

(2) Each voter individually blinds all of the messages (see Section 5.3) and sends them, with their blinding factors, to the CTF.

(3) The CTF checks its database to make sure the voter has not submitted his blinded votes for signature previously. It opens nine of the sets to check that they are properly formed. Then it individually signs each message in the set. It sends them back to the voter, storing the name of the voter in its database.

Election Systems & Software, LLC - Exhibit 1009
Election Systems & Software, LLC v. Hart Intercivic, Inc.
Page 9

Although the algorithm was one of the first public-key algorithms, and there were no successful cryptanalytic results against the algorithm, it has never gained wide acceptance in the cryptographic community. The scheme is two to three orders of magnitude faster than RSA, but has some problems. The public key is enormous: 2^{19} bits long. The data expansion is large: The ciphertext is twice as long as the plaintext.

Some attempts at cryptanalysis of this system can be found in [8,943,1559,306]. None of these were successful in the general case, although the similarity between the McEliece algorithm and knapsacks worried some.

In 1991, two Russian cryptographers claimed to have broken the McEliece system with some parameters [882]. Their paper contained no evidence to substantiate their claim, and most cryptographers discount the result. Another Russian attack, one that cannot be used directly against the McEliece system, is in [1447,1448]. Extensions to McEliece can be found in [424,1227,976].

Other Algorithms Based on Linear Error-Correcting Codes

The Niederreiter algorithm [1167] is closely related to the McEliece algorithm, and assumes that the public key is a random parity-check matrix of an error-correcting code. The private key is an efficient decoding algorithm for this matrix.

Another algorithm, used for identification and digital signatures, is based on syndrome decoding [1501]; see [306] for comments. An algorithm based on error-correcting codes [1621] is insecure [698,33,31,1560,32].

19.8 Elliptic Curve Cryptosystems

Elliptic curves have been studied for many years and there is an enormous amount of literature on the subject. In 1985, Neal Koblitz and V. S. Miller independently proposed using them for public-key cryptosystems [867,1095]. They did not invent a new cryptographic algorithm with elliptic curves over finite fields, but they implemented existing public-key algorithms, like Diffie-Hellman, using elliptic curves.

Elliptic curves are interesting because they provide a way of constructing "elements" and "rules of combining" that produce groups. These groups have enough familiar properties to build cryptographic algorithms, but they don't have certain properties that may facilitate cryptanalysis. For example, there is no good notion of "smooth" with elliptic curves. That is, there is no set of small elements in terms of which a random element has a good chance of being expressed by a simple algorithm. Hence, index calculus discrete logarithm algorithms do not work. See [1095] for more details.

Elliptic curves over the finite field $GF(2^n)$ are particularly interesting. The arithmetic processors for the underlying field are easy to construct and are relatively simple to implement for n in the range of 130 to 200. They have the potential to provide faster public-key cryptosystems with smaller key sizes. Many public-key algorithms, like Diffie-Hellman, ElGamal, and Schnorr, can be implemented in elliptic curves over finite fields.

The mathematics here are complex and beyond the scope of this book. Those interested in this topic are invited to read the two references previously mentioned,

Election Systems & Software, LLC - Exhibit 1009
Election Systems & Software, LLC v. Hart Intercivic, Inc.
Page 10

and the excellent book by Alfred Menezes [1059]. Two analogues of RSA work in elliptic curves [890,454]. Other papers are [23,119,1062,869,152,871,892,25,895,353,1061,26,913,914,915]. Elliptic curve cryptosystems with small key lengths are discussed in [701]. Next Computer Inc.'s Fast Elliptic Encryption (FEE) algorithm also uses elliptic curves [388]. FEE has the nice feature that the private key can be any easy-to-remember string. There are proposed public-key cryptosystems using hyperelliptic curves [868,870,1441,1214].

19.9 LUC

Some cryptographers have developed generalizations of RSA that use various permutation polynomials instead of exponentiation. A variation called Kravitz-Reed, using irreducible binary polynomials [898], is insecure [451,589]. Winfried Müller and Wilfried Nöbauer use Dickson polynomials [1127,1128,965]. Rudolph Lidl and Müller generalized this approach in [966,1126] (a variant is called the Réidi scheme), and Nöbauer looked at its security in [1172,1173]. (Comments on prime generation with Lucas functions are in [969,967,968,598].) Despite all of this prior art, a group of researchers from New Zealand managed to patent this scheme in 1993, calling it LUC [1486,521,1487].

The nth Lucas number, $V_n(P,1)$, is defined as

$$V_n(P,1) = PV_{n-1}(P,1) - V_{n-2}(P,1)$$

There's a lot more theory to Lucas numbers; I'm ignoring all of it. A good theoretical treatment of Lucas sequences is in [1307,1308]. A particularly nice description of the mathematics of LUC is in [1494,708].

In any case, to generate a public-key/private-key key pair, first choose two large primes, p and q. Calculate n, the product of p and q. The encryption key, e, is a random number that is relatively prime to $p-1$, $q-1$, $p+1$, and $q+1$.

There are four possible decryption keys,

$$d \equiv e^{-1} \bmod (\text{lcm}((p+1),(q+1)))$$
$$d \equiv e^{-1} \bmod (\text{lcm}((p+1),(q-1)))$$
$$d \equiv e^{-1} \bmod (\text{lcm}((p-1),(q+1)))$$
$$d \equiv e^{-1} \bmod (\text{lcm}((p-1),(q-1)))$$

where lcm is the least common multiple.

The public key is d and n; the private key is e and n. Discard p and q. To encrypt a message, P (P must be less than n), calculate

$$C \equiv V_e(P,1) \pmod n$$

And to decrypt:

$$P \equiv V_d(P,1) \pmod n, \text{ with the proper } d$$

At best, LUC is no more secure than RSA. And recent, still-unpublished results