

HANDBOOK of APPLIED CRYPTOGRAPHY

Alfred J. Menezes
Paul C. van Oorschot
Scott A. Vanstone



CRC Press

Boca Raton New York London Tokyo

Library of Congress Cataloging-in-Publication Data

Menezes, A. J. (Alfred J.), 1965-
Handbook of applied cryptography / Alfred Menezes, Paul van Oorschot,
Scott Vanstone.
p. cm. -- (CRC Press series on discrete mathematics and its
applications)
Includes bibliographical references and index.
ISBN 0-8493-8523-7 (alk. paper)
1. Computers--Access control--Handbooks, manuals, etc.
2. Cryptography--Handbooks, manuals, etc. I. Van Oorschot, Paul C.
II. Vanstone, Scott A. III. Title. IV. Series: Discrete
mathematics and its applications.
QA76.9.A25M463 1996
005.8'2--dc20

96-27609
CIP

This book contains information obtained from authentic and highly regarded sources. Reprinted material is quoted with permission, and sources are indicated. A wide variety of references are listed. Reasonable efforts have been made to publish reliable data and information, but the author and the publisher cannot assume responsibility for the validity of all materials or for the consequences of their use.

Neither this book nor any part may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, microfilming, and recording, or by any information storage or retrieval system, without prior permission in writing from the publisher.

The consent of CRC Press does not extend to copying for general distribution, for promotion, for creating new works, or for resale. Specific permission must be obtained in writing from CRC Press for such copying.

Direct all inquiries to CRC Press, Inc., 2000 Corporate Blvd., N.W., Boca Raton, Florida 33431.

© 1997 by CRC Press, Inc.

No claim to original U.S. Government works
International Standard Book Number 0-8493-8523-7
Library of Congress Card Number 96-27609
Printed in the United States of America 1 2 3 4 5 6 7 8 9 0
Printed on acid-free paper

To Archie and Lida Menezes

To Cornelis Henricus van Oorschot
and Maria Anna Buys van Vugt

To Margret and Gordon Vanstone

Contents in Brief

Table of Contents	v
List of Tables	xv
List of Figures	xix
Foreword	xxi
Preface	xxiii
1 Overview of Cryptography	1
2 Mathematical Background	49
3 Number-Theoretic Reference Problems	87
4 Public-Key Parameters	133
5 Pseudorandom Bits and Sequences	169
6 Stream Ciphers	191
7 Block Ciphers	223
8 Public-Key Encryption	283
9 Hash Functions and Data Integrity	321
10 Identification and Entity Authentication	385
11 Digital Signatures	425
12 Key Establishment Protocols	489
13 Key Management Techniques	543
14 Efficient Implementation	591
15 Patents and Standards	635
A Bibliography of Papers from Selected Cryptographic Forums	663
References	703
Index	755

Table of Contents

List of Tables	xv
List of Figures	xix
Foreword by R.L. Rivest	xxi
Preface	xxiii
1 Overview of Cryptography	1
1.1 Introduction	1
1.2 Information security and cryptography	2
1.3 Background on functions	6
1.3.1 Functions (1-1, one-way, trapdoor one-way)	6
1.3.2 Permutations	10
1.3.3 Involutions	10
1.4 Basic terminology and concepts	11
1.5 Symmetric-key encryption	15
1.5.1 Overview of block ciphers and stream ciphers	15
1.5.2 Substitution ciphers and transposition ciphers	17
1.5.3 Composition of ciphers	19
1.5.4 Stream ciphers	20
1.5.5 The key space	21
1.6 Digital signatures	22
1.7 Authentication and identification	24
1.7.1 Identification	24
1.7.2 Data origin authentication	25
1.8 Public-key cryptography	25
1.8.1 Public-key encryption	25
1.8.2 The necessity of authentication in public-key systems	27
1.8.3 Digital signatures from reversible public-key encryption	28
1.8.4 Symmetric-key vs. public-key cryptography	31
1.9 Hash functions	33
1.10 Protocols and mechanisms	33
1.11 Key establishment, management, and certification	35
1.11.1 Key management through symmetric-key techniques	36
1.11.2 Key management through public-key techniques	37
1.11.3 Trusted third parties and public-key certificates	39
1.12 Pseudorandom numbers and sequences	39
1.13 Classes of attacks and security models	41
1.13.1 Attacks on encryption schemes	41
1.13.2 Attacks on protocols	42
1.13.3 Models for evaluating security	42
1.13.4 Perspective for computational security	44
1.14 Notes and further references	45

2	Mathematical Background	49
2.1	Probability theory	50
2.1.1	Basic definitions	50
2.1.2	Conditional probability	51
2.1.3	Random variables	51
2.1.4	Binomial distribution	52
2.1.5	Birthday attacks	53
2.1.6	Random mappings	54
2.2	Information theory	56
2.2.1	Entropy	56
2.2.2	Mutual information	57
2.3	Complexity theory	57
2.3.1	Basic definitions	57
2.3.2	Asymptotic notation	58
2.3.3	Complexity classes	59
2.3.4	Randomized algorithms	62
2.4	Number theory	63
2.4.1	The integers	63
2.4.2	Algorithms in $\mathbb{Z}$	66
2.4.3	The integers modulo n	67
2.4.4	Algorithms in $\mathbb{Z}_n$	71
2.4.5	The Legendre and Jacobi symbols	72
2.4.6	Blum integers	74
2.5	Abstract algebra	75
2.5.1	Groups	75
2.5.2	Rings	76
2.5.3	Fields	77
2.5.4	Polynomial rings	78
2.5.5	Vector spaces	79
2.6	Finite fields	80
2.6.1	Basic properties	80
2.6.2	The Euclidean algorithm for polynomials	81
2.6.3	Arithmetic of polynomials	83
2.7	Notes and further references	85
3	Number-Theoretic Reference Problems	87
3.1	Introduction and overview	87
3.2	The integer factorization problem	89
3.2.1	Trial division	90
3.2.2	Pollard's rho factoring algorithm	91
3.2.3	Pollard's $p - 1$ factoring algorithm	92
3.2.4	Elliptic curve factoring	94
3.2.5	Random square factoring methods	94
3.2.6	Quadratic sieve factoring	95
3.2.7	Number field sieve factoring	98
3.3	The RSA problem	98
3.4	The quadratic residuosity problem	99
3.5	Computing square roots in $\mathbb{Z}_n$	99
3.5.1	Case (i): n prime	100
3.5.2	Case (ii): n composite	101

3.6	The discrete logarithm problem	103
3.6.1	Exhaustive search	104
3.6.2	Baby-step giant-step algorithm	104
3.6.3	Pollard's rho algorithm for logarithms	106
3.6.4	Pohlig-Hellman algorithm	107
3.6.5	Index-calculus algorithm	109
3.6.6	Discrete logarithm problem in subgroups of $\mathbb{Z}_p^*$	113
3.7	The Diffie-Hellman problem	113
3.8	Composite moduli	114
3.9	Computing individual bits	114
3.9.1	The discrete logarithm problem in $\mathbb{Z}_p^*$ — individual bits	116
3.9.2	The RSA problem — individual bits	116
3.9.3	The Rabin problem — individual bits	117
3.10	The subset sum problem	117
3.10.1	The L^3 -lattice basis reduction algorithm	118
3.10.2	Solving subset sum problems of low density	120
3.10.3	Simultaneous diophantine approximation	121
3.11	Factoring polynomials over finite fields	122
3.11.1	Square-free factorization	123
3.11.2	Berlekamp's Q -matrix algorithm	124
3.12	Notes and further references	125
4	Public-Key Parameters	133
4.1	Introduction	133
4.1.1	Generating large prime numbers naively	134
4.1.2	Distribution of prime numbers	134
4.2	Probabilistic primality tests	135
4.2.1	Fermat's test	136
4.2.2	Solovay-Strassen test	137
4.2.3	Miller-Rabin test	138
4.2.4	Comparison: Fermat, Solovay-Strassen, and Miller-Rabin	140
4.3	(True) Primality tests	142
4.3.1	Testing Mersenne numbers	142
4.3.2	Primality testing using the factorization of $n - 1$	143
4.3.3	Jacobi sum test	144
4.3.4	Tests using elliptic curves	145
4.4	Prime number generation	145
4.4.1	Random search for probable primes	145
4.4.2	Strong primes	149
4.4.3	NIST method for generating DSA primes	150
4.4.4	Constructive techniques for provable primes	152
4.5	Irreducible polynomials over $\mathbb{Z}_p$	154
4.5.1	Irreducible polynomials	154
4.5.2	Irreducible trinomials	157
4.5.3	Primitive polynomials	157
4.6	Generators and elements of high order	160
4.6.1	Selecting a prime p and generator of $\mathbb{Z}_p^*$	164
4.7	Notes and further references	165

5	Pseudorandom Bits and Sequences	169
5.1	Introduction	169
5.1.1	Background and Classification	170
5.2	Random bit generation	171
5.3	Pseudorandom bit generation	173
5.3.1	ANSI X9.17 generator	173
5.3.2	FIPS 186 generator	174
5.4	Statistical tests	175
5.4.1	The normal and chi-square distributions	176
5.4.2	Hypothesis testing	179
5.4.3	Golomb's randomness postulates	180
5.4.4	Five basic tests	181
5.4.5	Maurer's universal statistical test	183
5.5	Cryptographically secure pseudorandom bit generation	185
5.5.1	RSA pseudorandom bit generator	185
5.5.2	Blum-Blum-Shub pseudorandom bit generator	186
5.6	Notes and further references	187
6	Stream Ciphers	191
6.1	Introduction	191
6.1.1	Classification	192
6.2	Feedback shift registers	195
6.2.1	Linear feedback shift registers	195
6.2.2	Linear complexity	198
6.2.3	Berlekamp-Massey algorithm	200
6.2.4	Nonlinear feedback shift registers	202
6.3	Stream ciphers based on LFSRs	203
6.3.1	Nonlinear combination generators	205
6.3.2	Nonlinear filter generators	208
6.3.3	Clock-controlled generators	209
6.4	Other stream ciphers	212
6.4.1	SEAL	213
6.5	Notes and further references	216
7	Block Ciphers	223
7.1	Introduction and overview	223
7.2	Background and general concepts	224
7.2.1	Introduction to block ciphers	224
7.2.2	Modes of operation	228
7.2.3	Exhaustive key search and multiple encryption	233
7.3	Classical ciphers and historical development	237
7.3.1	Transposition ciphers (background)	238
7.3.2	Substitution ciphers (background)	238
7.3.3	Polyalphabetic substitutions and Vigenère ciphers (historical)	241
7.3.4	Polyalphabetic cipher machines and rotors (historical)	242
7.3.5	Cryptanalysis of classical ciphers (historical)	245
7.4	DES	250
7.4.1	Product ciphers and Feistel ciphers	250
7.4.2	DES algorithm	252
7.4.3	DES properties and strength	256

7.5	FEAL	259
7.6	IDEA	263
7.7	SAFER, RC5, and other block ciphers	266
7.7.1	SAFER	266
7.7.2	RC5	269
7.7.3	Other block ciphers	270
7.8	Notes and further references	271
8	Public-Key Encryption	283
8.1	Introduction	283
8.1.1	Basic principles	284
8.2	RSA public-key encryption	285
8.2.1	Description	286
8.2.2	Security of RSA	287
8.2.3	RSA encryption in practice	290
8.3	Rabin public-key encryption	292
8.4	ElGamal public-key encryption	294
8.4.1	Basic ElGamal encryption	294
8.4.2	Generalized ElGamal encryption	297
8.5	McEliece public-key encryption	298
8.6	Knapsack public-key encryption	300
8.6.1	Merkle-Hellman knapsack encryption	300
8.6.2	Chor-Rivest knapsack encryption	302
8.7	Probabilistic public-key encryption	306
8.7.1	Goldwasser-Micali probabilistic encryption	307
8.7.2	Blum-Goldwasser probabilistic encryption	308
8.7.3	Plaintext-aware encryption	311
8.8	Notes and further references	312
9	Hash Functions and Data Integrity	321
9.1	Introduction	321
9.2	Classification and framework	322
9.2.1	General classification	322
9.2.2	Basic properties and definitions	323
9.2.3	Hash properties required for specific applications	327
9.2.4	One-way functions and compression functions	327
9.2.5	Relationships between properties	329
9.2.6	Other hash function properties and applications	330
9.3	Basic constructions and general results	332
9.3.1	General model for iterated hash functions	332
9.3.2	General constructions and extensions	333
9.3.3	Formatting and initialization details	334
9.3.4	Security objectives and basic attacks	335
9.3.5	Bitsizes required for practical security	337
9.4	Unkeyed hash functions (MDCs)	338
9.4.1	Hash functions based on block ciphers	338
9.4.2	Customized hash functions based on MD4	343
9.4.3	Hash functions based on modular arithmetic	351
9.5	Keyed hash functions (MACs)	352
9.5.1	MACs based on block ciphers	353

9.5.2	Constructing MACs from MDCs	354
9.5.3	Customized MACs	356
9.5.4	MACs for stream ciphers	358
9.6	Data integrity and message authentication	359
9.6.1	Background and definitions	359
9.6.2	Non-malicious vs. malicious threats to data integrity	362
9.6.3	Data integrity using a MAC alone	364
9.6.4	Data integrity using an MDC and an authentic channel	364
9.6.5	Data integrity combined with encryption	364
9.7	Advanced attacks on hash functions	368
9.7.1	Birthday attacks	369
9.7.2	Pseudo-collisions and compression function attacks	371
9.7.3	Chaining attacks	373
9.7.4	Attacks based on properties of underlying cipher	375
9.8	Notes and further references	376
10	Identification and Entity Authentication	385
10.1	Introduction	385
10.1.1	Identification objectives and applications	386
10.1.2	Properties of identification protocols	387
10.2	Passwords (weak authentication)	388
10.2.1	Fixed password schemes: techniques	389
10.2.2	Fixed password schemes: attacks	391
10.2.3	Case study – UNIX passwords	393
10.2.4	PINs and passkeys	394
10.2.5	One-time passwords (towards strong authentication)	395
10.3	Challenge-response identification (strong authentication)	397
10.3.1	Background on time-variant parameters	397
10.3.2	Challenge-response by symmetric-key techniques	400
10.3.3	Challenge-response by public-key techniques	403
10.4	Customized and zero-knowledge identification protocols	405
10.4.1	Overview of zero-knowledge concepts	405
10.4.2	Feige-Fiat-Shamir identification protocol	410
10.4.3	GQ identification protocol	412
10.4.4	Schnorr identification protocol	414
10.4.5	Comparison: Fiat-Shamir, GQ, and Schnorr	416
10.5	Attacks on identification protocols	417
10.6	Notes and further references	420
11	Digital Signatures	425
11.1	Introduction	425
11.2	A framework for digital signature mechanisms	426
11.2.1	Basic definitions	426
11.2.2	Digital signature schemes with appendix	428
11.2.3	Digital signature schemes with message recovery	430
11.2.4	Types of attacks on signature schemes	432
11.3	RSA and related signature schemes	433
11.3.1	The RSA signature scheme	433
11.3.2	Possible attacks on RSA signatures	434
11.3.3	RSA signatures in practice	435

11.3.4	The Rabin public-key signature scheme	438
11.3.5	ISO/IEC 9796 formatting	442
11.3.6	PKCS #1 formatting	445
11.4	Fiat-Shamir signature schemes	447
11.4.1	Feige-Fiat-Shamir signature scheme	447
11.4.2	GQ signature scheme	450
11.5	The DSA and related signature schemes	451
11.5.1	The Digital Signature Algorithm (DSA)	452
11.5.2	The ElGamal signature scheme	454
11.5.3	The Schnorr signature scheme	459
11.5.4	The ElGamal signature scheme with message recovery	460
11.6	One-time digital signatures	462
11.6.1	The Rabin one-time signature scheme	462
11.6.2	The Merkle one-time signature scheme	464
11.6.3	Authentication trees and one-time signatures	466
11.6.4	The GMR one-time signature scheme	468
11.7	Other signature schemes	471
11.7.1	Arbitrated digital signatures	472
11.7.2	ESIGN	473
11.8	Signatures with additional functionality	474
11.8.1	Blind signature schemes	475
11.8.2	Undeniable signature schemes	476
11.8.3	Fail-stop signature schemes	478
11.9	Notes and further references	481
12	Key Establishment Protocols	489
12.1	Introduction	489
12.2	Classification and framework	490
12.2.1	General classification and fundamental concepts	490
12.2.2	Objectives and properties	493
12.2.3	Assumptions and adversaries in key establishment protocols	495
12.3	Key transport based on symmetric encryption	497
12.3.1	Symmetric key transport and derivation without a server	497
12.3.2	Kerberos and related server-based protocols	500
12.4	Key agreement based on symmetric techniques	505
12.5	Key transport based on public-key encryption	506
12.5.1	Key transport using PK encryption without signatures	507
12.5.2	Protocols combining PK encryption and signatures	509
12.5.3	Hybrid key transport protocols using PK encryption	512
12.6	Key agreement based on asymmetric techniques	515
12.6.1	Diffie-Hellman and related key agreement protocols	515
12.6.2	Implicitly-certified public keys	520
12.6.3	Diffie-Hellman protocols using implicitly-certified keys	522
12.7	Secret sharing	524
12.7.1	Simple shared control schemes	524
12.7.2	Threshold schemes	525
12.7.3	Generalized secret sharing	526
12.8	Conference keying	528
12.9	Analysis of key establishment protocols	530
12.9.1	Attack strategies and classic protocol flaws	530

12.9.2 Analysis objectives and methods	532
12.10 Notes and further references	534
13 Key Management Techniques	543
13.1 Introduction	543
13.2 Background and basic concepts	544
13.2.1 Classifying keys by algorithm type and intended use	544
13.2.2 Key management objectives, threats, and policy	545
13.2.3 Simple key establishment models	546
13.2.4 Roles of third parties	547
13.2.5 Tradeoffs among key establishment protocols	550
13.3 Techniques for distributing confidential keys	551
13.3.1 Key layering and cryptoperiods	551
13.3.2 Key translation centers and symmetric-key certificates	553
13.4 Techniques for distributing public keys	555
13.4.1 Authentication trees	556
13.4.2 Public-key certificates	559
13.4.3 Identity-based systems	561
13.4.4 Implicitly-certified public keys	562
13.4.5 Comparison of techniques for distributing public keys	563
13.5 Techniques for controlling key usage	567
13.5.1 Key separation and constraints on key usage	567
13.5.2 Techniques for controlling use of symmetric keys	568
13.6 Key management involving multiple domains	570
13.6.1 Trust between two domains	570
13.6.2 Trust models involving multiple certification authorities	572
13.6.3 Certificate distribution and revocation	576
13.7 Key life cycle issues	577
13.7.1 Lifetime protection requirements	578
13.7.2 Key management life cycle	578
13.8 Advanced trusted third party services	581
13.8.1 Trusted timestamping service	581
13.8.2 Non-repudiation and notarization of digital signatures	582
13.8.3 Key escrow	584
13.9 Notes and further references	586
14 Efficient Implementation	591
14.1 Introduction	591
14.2 Multiple-precision integer arithmetic	592
14.2.1 Radix representation	592
14.2.2 Addition and subtraction	594
14.2.3 Multiplication	595
14.2.4 Squaring	596
14.2.5 Division	598
14.3 Multiple-precision modular arithmetic	599
14.3.1 Classical modular multiplication	600
14.3.2 Montgomery reduction	600
14.3.3 Barrett reduction	603
14.3.4 Reduction methods for moduli of special form	605
14.4 Greatest common divisor algorithms	606

14.4.1	Binary gcd algorithm	606
14.4.2	Lehmer's gcd algorithm	607
14.4.3	Binary extended gcd algorithm	608
14.5	Chinese remainder theorem for integers	610
14.5.1	Residue number systems	611
14.5.2	Garner's algorithm	612
14.6	Exponentiation	613
14.6.1	Techniques for general exponentiation	614
14.6.2	Fixed-exponent exponentiation algorithms	620
14.6.3	Fixed-base exponentiation algorithms	623
14.7	Exponent recoding	627
14.7.1	Signed-digit representation	627
14.7.2	String-replacement representation	628
14.8	Notes and further references	630
15	Patents and Standards	635
15.1	Introduction	635
15.2	Patents on cryptographic techniques	635
15.2.1	Five fundamental patents	636
15.2.2	Ten prominent patents	638
15.2.3	Ten selected patents	641
15.2.4	Ordering and acquiring patents	645
15.3	Cryptographic standards	645
15.3.1	International standards – cryptographic techniques	645
15.3.2	Banking security standards (ANSI, ISO)	648
15.3.3	International security architectures and frameworks	653
15.3.4	U.S. government standards (FIPS)	654
15.3.5	Internet standards and RFCs	655
15.3.6	De facto standards	656
15.3.7	Ordering and acquiring standards	656
15.4	Notes and further references	657
A	Bibliography of Papers from Selected Cryptographic Forums	663
A.1	Asiacrypt/Auscrypt Proceedings	663
A.2	Crypto Proceedings	667
A.3	Eurocrypt Proceedings	684
A.4	Fast Software Encryption Proceedings	698
A.5	Journal of Cryptology papers	700
	References	703
	Index	755

This is a Chapter from the **Handbook of Applied Cryptography**, by A. Menezes, P. van Oorschot, and S. Vanstone, CRC Press, 1996.

For further information, see www.cacr.math.uwaterloo.ca/hac

CRC Press has granted the following specific permissions for the electronic version of this book:

Permission is granted to retrieve, print and store a single copy of this chapter for personal use. This permission does not extend to binding multiple chapters of the book, photocopying or producing copies for other than personal use of the person creating the copy, or making electronic copies available for retrieval by others without prior permission in writing from CRC Press.

Except where over-ridden by the specific permission above, the standard copyright notice from CRC Press applies to this electronic version:

Neither this book nor any part may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, microfilming, and recording, or by any information storage or retrieval system, without prior permission in writing from the publisher.

The consent of CRC Press does not extend to copying for general distribution, for promotion, for creating new works, or for resale. Specific permission must be obtained in writing from CRC Press for such copying.

©1997 by CRC Press, Inc.

Chapter 1

Overview of Cryptography

Contents in Brief

1.1	Introduction	1
1.2	Information security and cryptography	2
1.3	Background on functions	6
1.4	Basic terminology and concepts	11
1.5	Symmetric-key encryption	15
1.6	Digital signatures	22
1.7	Authentication and identification	24
1.8	Public-key cryptography	25
1.9	Hash functions	33
1.10	Protocols and mechanisms	33
1.11	Key establishment, management, and certification	35
1.12	Pseudorandom numbers and sequences	39
1.13	Classes of attacks and security models	41
1.14	Notes and further references	45

1.1 Introduction

Cryptography has a long and fascinating history. The most complete non-technical account of the subject is Kahn's *The Codebreakers*. This book traces cryptography from its initial and limited use by the Egyptians some 4000 years ago, to the twentieth century where it played a crucial role in the outcome of both world wars. Completed in 1963, Kahn's book covers those aspects of the history which were most significant (up to that time) to the development of the subject. The predominant practitioners of the art were those associated with the military, the diplomatic service and government in general. Cryptography was used as a tool to protect national secrets and strategies.

The proliferation of computers and communications systems in the 1960s brought with it a demand from the private sector for means to protect information in digital form and to provide security services. Beginning with the work of Feistel at IBM in the early 1970s and culminating in 1977 with the adoption as a U.S. Federal Information Processing Standard for encrypting unclassified information, DES, the Data Encryption Standard, is the most well-known cryptographic mechanism in history. It remains the standard means for securing electronic commerce for many financial institutions around the world.

The most striking development in the history of cryptography came in 1976 when Diffie and Hellman published *New Directions in Cryptography*. This paper introduced the revolutionary concept of public-key cryptography and also provided a new and ingenious method

for key exchange, the security of which is based on the intractability of the discrete logarithm problem. Although the authors had no practical realization of a public-key encryption scheme at the time, the idea was clear and it generated extensive interest and activity in the cryptographic community. In 1978 Rivest, Shamir, and Adleman discovered the first practical public-key encryption and signature scheme, now referred to as RSA. The RSA scheme is based on another hard mathematical problem, the intractability of factoring large integers. This application of a hard mathematical problem to cryptography revitalized efforts to find more efficient methods to factor. The 1980s saw major advances in this area but none which rendered the RSA system insecure. Another class of powerful and practical public-key schemes was found by ElGamal in 1985. These are also based on the discrete logarithm problem.

One of the most significant contributions provided by public-key cryptography is the digital signature. In 1991 the first international standard for digital signatures (ISO/IEC 9796) was adopted. It is based on the RSA public-key scheme. In 1994 the U.S. Government adopted the Digital Signature Standard, a mechanism based on the ElGamal public-key scheme.

The search for new public-key schemes, improvements to existing cryptographic mechanisms, and proofs of security continues at a rapid pace. Various standards and infrastructures involving cryptography are being put in place. Security products are being developed to address the security needs of an information intensive society.

The purpose of this book is to give an up-to-date treatise of the principles, techniques, and algorithms of interest in cryptographic practice. Emphasis has been placed on those aspects which are most practical and applied. The reader will be made aware of the basic issues and pointed to specific related research in the literature where more indepth discussions can be found. Due to the volume of material which is covered, most results will be stated without proofs. This also serves the purpose of not obscuring the very applied nature of the subject. This book is intended for both implementers and researchers. It describes algorithms, systems, and their interactions.

Chapter 1 is a tutorial on the many and various aspects of cryptography. It does not attempt to convey all of the details and subtleties inherent to the subject. Its purpose is to introduce the basic issues and principles and to point the reader to appropriate chapters in the book for more comprehensive treatments. Specific techniques are avoided in this chapter.

1.2 Information security and cryptography

The concept of *information* will be taken to be an understood quantity. To introduce cryptography, an understanding of issues related to information security in general is necessary. Information security manifests itself in many ways according to the situation and requirement. Regardless of who is involved, to one degree or another, all parties to a transaction must have confidence that certain objectives associated with information security have been met. Some of these objectives are listed in Table 1.1.

Over the centuries, an elaborate set of protocols and mechanisms has been created to deal with information security issues when the information is conveyed by physical documents. Often the objectives of information security cannot solely be achieved through mathematical algorithms and protocols alone, but require procedural techniques and abidance of laws to achieve the desired result. For example, privacy of letters is provided by sealed envelopes delivered by an accepted mail service. The physical security of the envelope is, for practical necessity, limited and so laws are enacted which make it a criminal

©1997 by CRC Press, Inc. — See accompanying notice at front of chapter.

privacy or confidentiality	keeping information secret from all but those who are authorized to see it.
data integrity	ensuring information has not been altered by unauthorized or unknown means.
entity authentication or identification	corroboration of the identity of an entity (e.g., a person, a computer terminal, a credit card, etc.).
message authentication	corroborating the source of information; also known as data origin authentication.
signature	a means to bind information to an entity.
authorization	conveyance, to another entity, of official sanction to do or be something.
validation	a means to provide timeliness of authorization to use or manipulate information or resources.
access control	restricting access to resources to privileged entities.
certification	endorsement of information by a trusted entity.
timestamping	recording the time of creation or existence of information.
witnessing	verifying the creation or existence of information by an entity other than the creator.
receipt	acknowledgement that information has been received.
confirmation	acknowledgement that services have been provided.
ownership	a means to provide an entity with the legal right to use or transfer a resource to others.
anonymity	concealing the identity of an entity involved in some process.
non-repudiation	preventing the denial of previous commitments or actions.
revocation	retraction of certification or authorization.

Table 1.1: Some information security objectives.

offense to open mail for which one is not authorized. It is sometimes the case that security is achieved not through the information itself but through the physical document recording it. For example, paper currency requires special inks and material to prevent counterfeiting.

Conceptually, the way information is recorded has not changed dramatically over time. Whereas information was typically stored and transmitted on paper, much of it now resides on magnetic media and is transmitted via telecommunications systems, some wireless. What has changed dramatically is the ability to copy and alter information. One can make thousands of identical copies of a piece of information stored electronically and each is indistinguishable from the original. With information on paper, this is much more difficult. What is needed then for a society where information is mostly stored and transmitted in electronic form is a means to ensure information security which is independent of the physical medium recording or conveying it and such that the objectives of information security rely solely on digital information itself.

One of the fundamental tools used in information security is the signature. It is a building block for many other services such as non-repudiation, data origin authentication, identification, and witnessing, to mention a few. Having learned the basics in writing, an individual is taught how to produce a handwritten signature for the purpose of identification. At contract age the signature evolves to take on a very integral part of the person's identity. This signature is intended to be unique to the individual and serve as a means to identify, authorize, and validate. With electronic information the concept of a signature needs to be

redressed; it cannot simply be something unique to the signer and independent of the information signed. Electronic replication of it is so simple that appending a signature to a document not signed by the originator of the signature is almost a triviality.

Analogues of the “paper protocols” currently in use are required. Hopefully these new electronic based protocols are at least as good as those they replace. There is a unique opportunity for society to introduce new and more efficient ways of ensuring information security. Much can be learned from the evolution of the paper based system, mimicking those aspects which have served us well and removing the inefficiencies.

Achieving information security in an electronic society requires a vast array of technical and legal skills. There is, however, no guarantee that all of the information security objectives deemed necessary can be adequately met. The technical means is provided through cryptography.

1.1 Definition *Cryptography* is the study of mathematical techniques related to aspects of information security such as confidentiality, data integrity, entity authentication, and data origin authentication.

Cryptography is not the only means of providing information security, but rather one set of techniques.

Cryptographic goals

Of all the information security objectives listed in Table 1.1, the following four form a framework upon which the others will be derived: (1) privacy or confidentiality (§1.5, §1.8); (2) data integrity (§1.9); (3) authentication (§1.7); and (4) non-repudiation (§1.6).

1. *Confidentiality* is a service used to keep the content of information from all but those authorized to have it. *Secrecy* is a term synonymous with confidentiality and privacy. There are numerous approaches to providing confidentiality, ranging from physical protection to mathematical algorithms which render data unintelligible.
2. *Data integrity* is a service which addresses the unauthorized alteration of data. To assure data integrity, one must have the ability to detect data manipulation by unauthorized parties. Data manipulation includes such things as insertion, deletion, and substitution.
3. *Authentication* is a service related to identification. This function applies to both entities and information itself. Two parties entering into a communication should identify each other. Information delivered over a channel should be authenticated as to origin, date of origin, data content, time sent, etc. For these reasons this aspect of cryptography is usually subdivided into two major classes: *entity authentication* and *data origin authentication*. Data origin authentication implicitly provides data integrity (for if a message is modified, the source has changed).
4. *Non-repudiation* is a service which prevents an entity from denying previous commitments or actions. When disputes arise due to an entity denying that certain actions were taken, a means to resolve the situation is necessary. For example, one entity may authorize the purchase of property by another entity and later deny such authorization was granted. A procedure involving a trusted third party is needed to resolve the dispute.

A fundamental goal of cryptography is to adequately address these four areas in both theory and practice. Cryptography is about the prevention and detection of cheating and other malicious activities.

This book describes a number of basic *cryptographic tools (primitives)* used to provide information security. Examples of primitives include encryption schemes (§1.5 and §1.8),

©1997 by CRC Press, Inc. — See accompanying notice at front of chapter.

hash functions (§1.9), and digital signature schemes (§1.6). Figure 1.1 provides a schematic listing of the primitives considered and how they relate. Many of these will be briefly introduced in this chapter, with detailed discussion left to later chapters. These primitives should

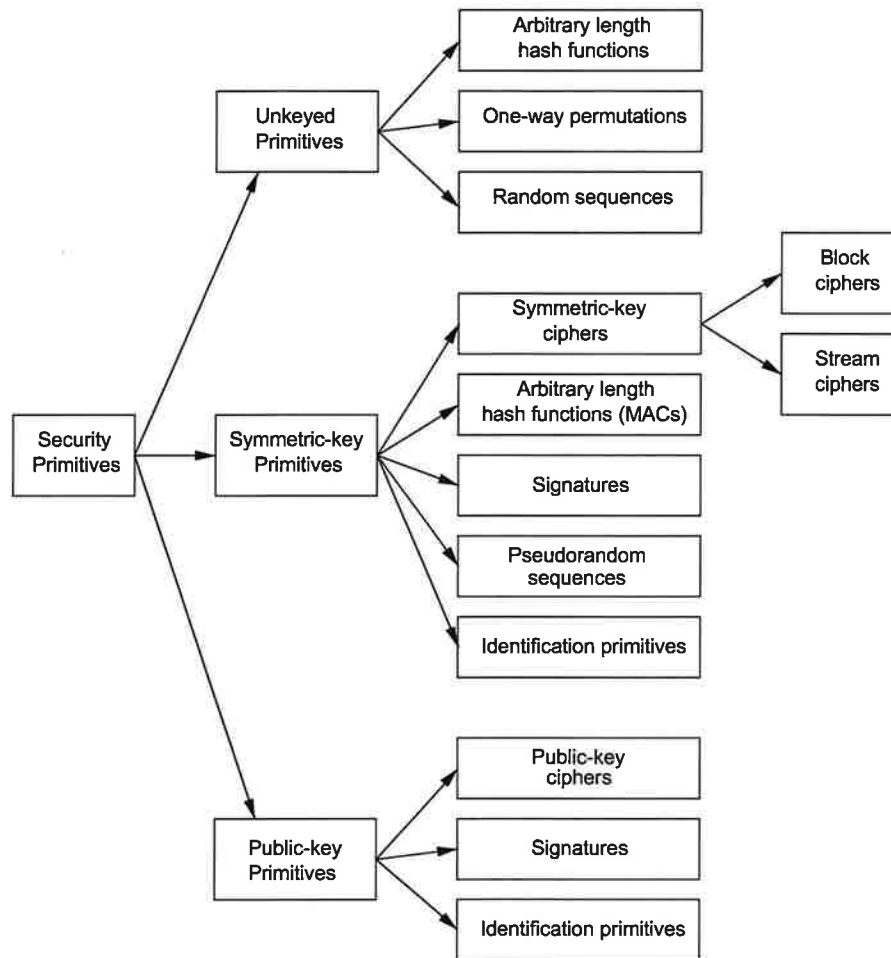


Figure 1.1: A taxonomy of cryptographic primitives.

be evaluated with respect to various criteria such as:

1. *level of security.* This is usually difficult to quantify. Often it is given in terms of the number of operations required (using the best methods currently known) to defeat the intended objective. Typically the level of security is defined by an upper bound on the amount of work necessary to defeat the objective. This is sometimes called the work factor (see §1.13.4).
2. *functionality.* Primitives will need to be combined to meet various information security objectives. Which primitives are most effective for a given objective will be determined by the basic properties of the primitives.
3. *methods of operation.* Primitives, when applied in various ways and with various inputs, will typically exhibit different characteristics; thus, one primitive could provide

Handbook of Applied Cryptography by A. Menezes, P. van Oorschot and S. Vanstone.

very different functionality depending on its mode of operation or usage.

4. *performance*. This refers to the efficiency of a primitive in a particular mode of operation. (For example, an encryption algorithm may be rated by the number of bits per second which it can encrypt.)
5. *ease of implementation*. This refers to the difficulty of realizing the primitive in a practical instantiation. This might include the complexity of implementing the primitive in either a software or hardware environment.

The relative importance of various criteria is very much dependent on the application and resources available. For example, in an environment where computing power is limited one may have to trade off a very high level of security for better performance of the system as a whole.

Cryptography, over the ages, has been an art practised by many who have devised ad hoc techniques to meet some of the information security requirements. The last twenty years have been a period of transition as the discipline moved from an art to a science. There are now several international scientific conferences devoted exclusively to cryptography and also an international scientific organization, the International Association for Cryptologic Research (IACR), aimed at fostering research in the area.

This book is about cryptography: the theory, the practice, and the standards.

1.3 Background on functions

While this book is not a treatise on abstract mathematics, a familiarity with basic mathematical concepts will prove to be useful. One concept which is absolutely fundamental to cryptography is that of a *function* in the mathematical sense. A function is alternately referred to as a *mapping* or a *transformation*.

1.3.1 Functions (1-1, one-way, trapdoor one-way)

A *set* consists of distinct objects which are called *elements* of the set. For example, a set X might consist of the elements a, b, c , and this is denoted $X = \{a, b, c\}$.

- 1.2 Definition** A *function* is defined by two sets X and Y and a *rule* f which assigns to each element in X precisely one element in Y . The set X is called the *domain* of the function and Y the *codomain*. If x is an element of X (usually written $x \in X$) the *image* of x is the element in Y which the rule f associates with x ; the image y of x is denoted by $y = f(x)$. Standard notation for a function f from set X to set Y is $f: X \rightarrow Y$. If $y \in Y$, then a *preimage* of y is an element $x \in X$ for which $f(x) = y$. The set of all elements in Y which have at least one preimage is called the *image* of f , denoted $\text{Im}(f)$.

- 1.3 Example (function)** Consider the sets $X = \{a, b, c\}$, $Y = \{1, 2, 3, 4\}$, and the rule f from X to Y defined as $f(a) = 2$, $f(b) = 4$, $f(c) = 1$. Figure 1.2 shows a schematic of the sets X , Y and the function f . The preimage of the element 2 is a . The image of f is $\{1, 2, 4\}$. $\square$

Thinking of a function in terms of the schematic (sometimes called a *functional diagram*) given in Figure 1.2, each element in the domain X has precisely one arrowed line originating from it. Each element in the codomain Y can have any number of arrowed lines incident to it (including zero lines).

©1997 by CRC Press, Inc. — See accompanying notice at front of chapter.

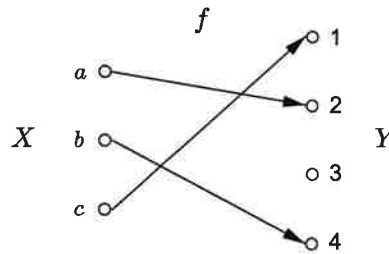


Figure 1.2: A function f from a set X of three elements to a set Y of four elements.

Often only the domain X and the rule f are given and the codomain is assumed to be the image of f . This point is illustrated with two examples.

1.4 Example (function) Take $X = \{1, 2, 3, \dots, 10\}$ and let f be the rule that for each $x \in X$, $f(x) = r_x$, where r_x is the remainder when x^2 is divided by 11. Explicitly then

$$\begin{array}{cccccc} f(1) = 1 & f(2) = 4 & f(3) = 9 & f(4) = 5 & f(5) = 3 \\ f(6) = 3 & f(7) = 5 & f(8) = 9 & f(9) = 4 & f(10) = 1. \end{array}$$

The image of f is the set $Y = \{1, 3, 4, 5, 9\}$. □

1.5 Example (function) Take $X = \{1, 2, 3, \dots, 10^{50}\}$ and let f be the rule $f(x) = r_x$, where r_x is the remainder when x^2 is divided by $10^{50} + 1$ for all $x \in X$. Here it is not feasible to write down f explicitly as in Example 1.4, but nonetheless the function is completely specified by the domain and the mathematical description of the rule f . □

(i) 1-1 functions

1.6 Definition A function (or transformation) is 1 – 1 (*one-to-one*) if each element in the codomain Y is the image of at most one element in the domain X .

1.7 Definition A function (or transformation) is *onto* if each element in the codomain Y is the image of at least one element in the domain. Equivalently, a function $f: X \rightarrow Y$ is onto if $\text{Im}(f) = Y$.

1.8 Definition If a function $f: X \rightarrow Y$ is 1 – 1 and $\text{Im}(f) = Y$, then f is called a *bijection*.

1.9 Fact If $f: X \rightarrow Y$ is 1 – 1 then $f: X \rightarrow \text{Im}(f)$ is a bijection. In particular, if $f: X \rightarrow Y$ is 1 – 1, and X and Y are finite sets of the same size, then f is a bijection.

In terms of the schematic representation, if f is a bijection, then each element in Y has exactly one arrowed line incident with it. The functions described in Examples 1.3 and 1.4 are not bijections. In Example 1.3 the element 3 is not the image of any element in the domain. In Example 1.4 each element in the codomain has two preimages.

1.10 Definition If f is a bijection from X to Y then it is a simple matter to define a bijection g from Y to X as follows: for each $y \in Y$ define $g(y) = x$ where $x \in X$ and $f(x) = y$. This function g obtained from f is called the *inverse function* of f and is denoted by $g = f^{-1}$.



Figure 1.3: A bijection f and its inverse $g = f^{-1}$.

1.11 Example (inverse function) Let $X = \{a, b, c, d, e\}$, and $Y = \{1, 2, 3, 4, 5\}$, and consider the rule f given by the arrowed edges in Figure 1.3. f is a bijection and its inverse g is formed simply by reversing the arrows on the edges. The domain of g is Y and the codomain is X . $\square$

Note that if f is a bijection, then so is f^{-1} . In cryptography bijections are used as the tool for encrypting messages and the inverse transformations are used to decrypt. This will be made clearer in §1.4 when some basic terminology is introduced. Notice that if the transformations were not bijections then it would not be possible to always decrypt to a unique message.

(ii) One-way functions

There are certain types of functions which play significant roles in cryptography. At the expense of rigor, an intuitive definition of a one-way function is given.

1.12 Definition A function f from a set X to a set Y is called a *one-way function* if $f(x)$ is “easy” to compute for all $x \in X$ but for “essentially all” elements $y \in \text{Im}(f)$ it is “computationally infeasible” to find any $x \in X$ such that $f(x) = y$.

1.13 Note (clarification of terms in Definition 1.12)

- (i) A rigorous definition of the terms “easy” and “computationally infeasible” is necessary but would detract from the simple idea that is being conveyed. For the purpose of this chapter, the intuitive meaning will suffice.
- (ii) The phrase “for essentially all elements in Y ” refers to the fact that there are a few values $y \in Y$ for which it is easy to find an $x \in X$ such that $y = f(x)$. For example, one may compute $y = f(x)$ for a small number of x values and then for these, the inverse is known by table look-up. An alternate way to describe this property of a one-way function is the following: for a random $y \in \text{Im}(f)$ it is computationally infeasible to find any $x \in X$ such that $f(x) = y$.

The concept of a one-way function is illustrated through the following examples.

1.14 Example (one-way function) Take $X = \{1, 2, 3, \dots, 16\}$ and define $f(x) = r_x$ for all $x \in X$ where r_x is the remainder when 3^x is divided by 17. Explicitly,

x	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$f(x)$	3	9	10	13	5	15	11	16	14	8	7	4	12	2	6	1

Given a number between 1 and 16, it is relatively easy to find the image of it under f . However, given a number such as 7, without having the table in front of you, it is harder to find

x given that $f(x) = 7$. Of course, if the number you are given is 3 then it is clear that $x = 1$ is what you need; but for most of the elements in the codomain it is not that easy. $\square$

One must keep in mind that this is an example which uses very small numbers; the important point here is that there is a difference in the amount of work to compute $f(x)$ and the amount of work to find x given $f(x)$. Even for very large numbers, $f(x)$ can be computed efficiently using the repeated square-and-multiply algorithm (Algorithm 2.143), whereas the process of finding x from $f(x)$ is much harder.

1.15 Example (one-way function) A *prime number* is a positive integer greater than 1 whose only positive integer divisors are 1 and itself. Select primes $p = 48611$, $q = 53993$, form $n = pq = 2624653723$, and let $X = \{1, 2, 3, \dots, n-1\}$. Define a function f on X by $f(x) = r_x$ for each $x \in X$, where r_x is the remainder when x^3 is divided by n . For instance, $f(2489991) = 1981394214$ since $2489991^3 = 5881949859 \cdot n + 1981394214$. Computing $f(x)$ is a relatively simple thing to do, but to reverse the procedure is much more difficult; that is, given a remainder to find the value x which was originally cubed (raised to the third power). This procedure is referred to as the computation of a modular cube root with modulus n . If the factors of n are unknown and large, this is a difficult problem; however, if the factors p and q of n are known then there is an efficient algorithm for computing modular cube roots. (See §8.2.2(i) for details.) $\square$

Example 1.15 leads one to consider another type of function which will prove to be fundamental in later developments.

(iii) Trapdoor one-way functions

1.16 Definition A *trapdoor one-way function* is a one-way function $f: X \rightarrow Y$ with the additional property that given some extra information (called the *trapdoor information*) it becomes feasible to find for any given $y \in \text{Im}(f)$, an $x \in X$ such that $f(x) = y$.

Example 1.15 illustrates the concept of a trapdoor one-way function. With the additional information of the factors of $n = 2624653723$ (namely, $p = 48611$ and $q = 53993$, each of which is five decimal digits long) it becomes much easier to invert the function. The factors of 2624653723 are large enough that finding them by hand computation would be difficult. Of course, any reasonable computer program could find the factors relatively quickly. If, on the other hand, one selects p and q to be very large distinct prime numbers (each having about 100 decimal digits) then, by today's standards, it is a difficult problem, even with the most powerful computers, to deduce p and q simply from n . This is the well-known *integer factorization problem* (see §3.2) and a source of many trapdoor one-way functions.

It remains to be rigorously established whether there actually are any (true) one-way functions. That is to say, no one has yet definitively proved the existence of such functions under reasonable (and rigorous) definitions of “easy” and “computationally infeasible”. Since the existence of one-way functions is still unknown, the existence of trapdoor one-way functions is also unknown. However, there are a number of good candidates for one-way and trapdoor one-way functions. Many of these are discussed in this book, with emphasis given to those which are practical.

One-way and trapdoor one-way functions are the basis for public-key cryptography (discussed in §1.8). The importance of these concepts will become clearer when their application to cryptographic techniques is considered. It will be worthwhile to keep the abstract concepts of this section in mind as concrete methods are presented.

1.3.2 Permutations

Permutations are functions which are often used in various cryptographic constructs.

1.17 Definition Let S be a finite set of elements. A *permutation* p on S is a bijection (Definition 1.8) from S to itself (i.e., $p: S \rightarrow S$).

1.18 Example (permutation) Let $S = \{1, 2, 3, 4, 5\}$. A permutation $p: S \rightarrow S$ is defined as follows:

$$p(1) = 3, p(2) = 5, p(3) = 4, p(4) = 2, p(5) = 1.$$

A permutation can be described in various ways. It can be displayed as above or as an array:

$$p = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 5 & 4 & 2 & 1 \end{pmatrix}, \quad (1.1)$$

where the top row in the array is the domain and the bottom row is the image under the mapping p . Of course, other representations are possible. $\square$

Since permutations are bijections, they have inverses. If a permutation is written as an array (see 1.1), its inverse is easily found by interchanging the rows in the array and reordering the elements in the new top row if desired (the bottom row would have to be reordered correspondingly). The inverse of p in Example 1.18 is $p^{-1} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 4 & 1 & 3 & 2 \end{pmatrix}$.

1.19 Example (permutation) Let X be the set of integers $\{0, 1, 2, \dots, pq - 1\}$ where p and q are distinct *large* primes (for example, p and q are each about 100 decimal digits long), and suppose that neither $p - 1$ nor $q - 1$ is divisible by 3. Then the function $p(x) = r_x$, where r_x is the remainder when x^3 is divided by pq , can be shown to be a permutation. Determining the inverse permutation is computationally infeasible by today's standards unless p and q are known (cf. Example 1.15). $\square$

1.3.3 Involutions

Another type of function which will be referred to in §1.5.3 is an involution. Involutions have the property that they are their own inverses.

1.20 Definition Let S be a finite set and let f be a bijection from S to S (i.e., $f: S \rightarrow S$). The function f is called an *involution* if $f = f^{-1}$. An equivalent way of stating this is $f(f(x)) = x$ for all $x \in S$.

1.21 Example (involution) Figure 1.4 is an example of an involution. In the diagram of an involution, note that if j is the image of i then i is the image of j . $\square$

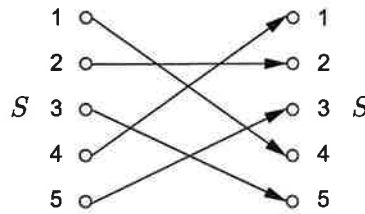


Figure 1.4: An involution on a set S of 5 elements.

1.4 Basic terminology and concepts

The scientific study of any discipline must be built upon rigorous definitions arising from fundamental concepts. What follows is a list of terms and basic concepts used throughout this book. Where appropriate, rigor has been sacrificed (here in Chapter 1) for the sake of clarity.

Encryption domains and codomains

- $\mathcal{A}$ denotes a finite set called the *alphabet of definition*. For example, $\mathcal{A} = \{0, 1\}$, the *binary alphabet*, is a frequently used alphabet of definition. Note that any alphabet can be encoded in terms of the binary alphabet. For example, since there are 32 binary strings of length five, each letter of the English alphabet can be assigned a unique binary string of length five.
- $\mathcal{M}$ denotes a set called the *message space*. $\mathcal{M}$ consists of strings of symbols from an alphabet of definition. An element of $\mathcal{M}$ is called a *plaintext message* or simply a *plaintext*. For example, $\mathcal{M}$ may consist of binary strings, English text, computer code, etc.
- $\mathcal{C}$ denotes a set called the *ciphertext space*. $\mathcal{C}$ consists of strings of symbols from an alphabet of definition, which may differ from the alphabet of definition for $\mathcal{M}$. An element of $\mathcal{C}$ is called a *ciphertext*.

Encryption and decryption transformations

- $\mathcal{K}$ denotes a set called the *key space*. An element of $\mathcal{K}$ is called a *key*.
- Each element $e \in \mathcal{K}$ uniquely determines a bijection from $\mathcal{M}$ to $\mathcal{C}$, denoted by E_e . E_e is called an *encryption function* or an *encryption transformation*. Note that E_e must be a bijection if the process is to be reversed and a unique plaintext message recovered for each distinct ciphertext.¹
- For each $d \in \mathcal{K}$, D_d denotes a bijection from $\mathcal{C}$ to $\mathcal{M}$ (i.e., $D_d: \mathcal{C} \rightarrow \mathcal{M}$). D_d is called a *decryption function* or *decryption transformation*.
- The process of applying the transformation E_e to a message $m \in \mathcal{M}$ is usually referred to as *encrypting m* or the *encryption of m* .
- The process of applying the transformation D_d to a ciphertext c is usually referred to as *decrypting c* or the *decryption of c* .

¹More generality is obtained if E_e is simply defined as a $1-1$ transformation from $\mathcal{M}$ to $\mathcal{C}$. That is to say, E_e is a bijection from $\mathcal{M}$ to $\text{Im}(E_e)$ where $\text{Im}(E_e)$ is a subset of $\mathcal{C}$.

- An *encryption scheme* consists of a set $\{E_e: e \in \mathcal{K}\}$ of encryption transformations and a corresponding set $\{D_d: d \in \mathcal{K}\}$ of decryption transformations with the property that for each $e \in \mathcal{K}$ there is a unique key $d \in \mathcal{K}$ such that $D_d = E_e^{-1}$; that is, $D_d(E_e(m)) = m$ for all $m \in \mathcal{M}$. An encryption scheme is sometimes referred to as a *cipher*.
- The keys e and d in the preceding definition are referred to as a *key pair* and sometimes denoted by (e, d) . Note that e and d could be the same.
- To *construct* an encryption scheme requires one to select a message space $\mathcal{M}$, a ciphertext space $\mathcal{C}$, a key space $\mathcal{K}$, a set of encryption transformations $\{E_e: e \in \mathcal{K}\}$, and a corresponding set of decryption transformations $\{D_d: d \in \mathcal{K}\}$.

Achieving confidentiality

An encryption scheme may be used as follows for the purpose of achieving confidentiality. Two parties Alice and Bob first secretly choose or secretly exchange a key pair (e, d) . At a subsequent point in time, if Alice wishes to send a message $m \in \mathcal{M}$ to Bob, she computes $c = E_e(m)$ and transmits this to Bob. Upon receiving c , Bob computes $D_d(c) = m$ and hence recovers the original message m .

The question arises as to why keys are necessary. (Why not just choose one encryption function and its corresponding decryption function?) Having transformations which are very similar but characterized by keys means that if some particular encryption/decryption transformation is revealed then one does not have to redesign the entire scheme but simply change the key. It is sound cryptographic practice to change the key (encryption/decryption transformation) frequently. As a physical analogue, consider an ordinary resettable combination lock. The structure of the lock is available to anyone who wishes to purchase one but the combination is chosen and set by the owner. If the owner suspects that the combination has been revealed he can easily reset it without replacing the physical mechanism.

1.22 Example (encryption scheme) Let $\mathcal{M} = \{m_1, m_2, m_3\}$ and $\mathcal{C} = \{c_1, c_2, c_3\}$. There are precisely $3! = 6$ bijections from $\mathcal{M}$ to $\mathcal{C}$. The key space $\mathcal{K} = \{1, 2, 3, 4, 5, 6\}$ has six elements in it, each specifying one of the transformations. Figure 1.5 illustrates the six encryption functions which are denoted by $E_i, 1 \leq i \leq 6$. Alice and Bob agree on a trans-

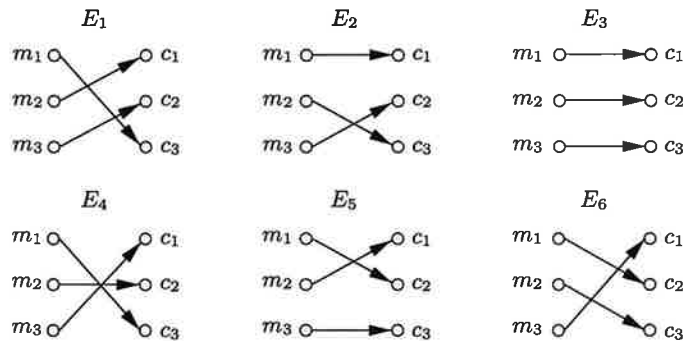


Figure 1.5: Schematic of a simple encryption scheme.

formation, say E_1 . To encrypt the message m_1 , Alice computes $E_1(m_1) = c_3$ and sends c_3 to Bob. Bob decrypts c_3 by reversing the arrows on the diagram for E_1 and observing that c_3 points to m_1 .

When $\mathcal{M}$ is a small set, the functional diagram is a simple visual means to describe the mapping. In cryptography, the set $\mathcal{M}$ is typically of astronomical proportions and, as such, the visual description is infeasible. What is required, in these cases, is some other simple means to describe the encryption and decryption transformations, such as mathematical algorithms. $\square$

Figure 1.6 provides a simple model of a two-party communication using encryption.

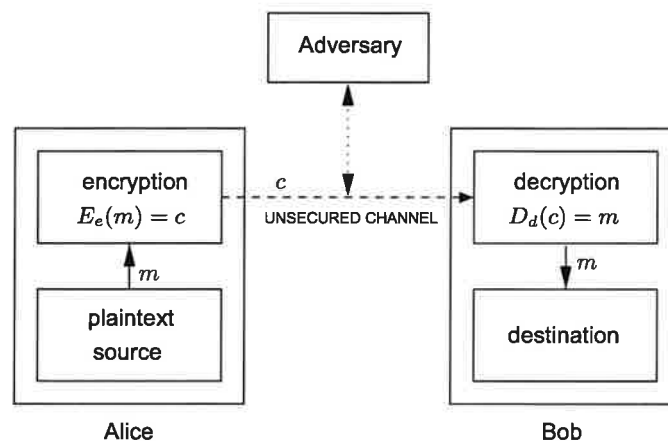


Figure 1.6: Schematic of a two-party communication using encryption.

Communication participants

Referring to Figure 1.6, the following terminology is defined.

- An *entity* or *party* is someone or something which sends, receives, or manipulates information. Alice and Bob are entities in Example 1.22. An entity may be a person, a computer terminal, etc.
- A *sender* is an entity in a two-party communication which is the legitimate transmitter of information. In Figure 1.6, the sender is Alice.
- A *receiver* is an entity in a two-party communication which is the intended recipient of information. In Figure 1.6, the receiver is Bob.
- An *adversary* is an entity in a two-party communication which is neither the sender nor receiver, and which tries to defeat the information security service being provided between the sender and receiver. Various other names are synonymous with adversary such as enemy, attacker, opponent, tapper, eavesdropper, intruder, and interloper. An adversary will often attempt to play the role of either the legitimate sender or the legitimate receiver.

Channels

- A *channel* is a means of conveying information from one entity to another.
- A *physically secure channel* or *secure channel* is one which is not physically accessible to the adversary.
- An *unsecured channel* is one from which parties other than those for which the information is intended can reorder, delete, insert, or read.
- A *secured channel* is one from which an adversary does not have the ability to reorder, delete, insert, or read.

Handbook of Applied Cryptography by A. Menezes, P. van Oorschot and S. Vanstone.

One should note the subtle difference between a physically secure channel and a secured channel – a secured channel may be secured by physical or cryptographic techniques, the latter being the topic of this book. Certain channels are assumed to be physically secure. These include trusted couriers, personal contact between communicating parties, and a dedicated communication link, to name a few.

Security

A fundamental premise in cryptography is that the sets $\mathcal{M}, \mathcal{C}, \mathcal{K}, \{E_e: e \in \mathcal{K}\}, \{D_d: d \in \mathcal{K}\}$ are public knowledge. When two parties wish to communicate securely using an encryption scheme, the only thing that they keep secret is the particular key pair (e, d) which they are using, and which they must select. One can gain additional security by keeping the class of encryption and decryption transformations secret but one should not base the security of the entire scheme on this approach. History has shown that maintaining the secrecy of the transformations is very difficult indeed.

1.23 Definition An encryption scheme is said to be *breakable* if a third party, without prior knowledge of the key pair (e, d) , can systematically recover plaintext from corresponding ciphertext within some appropriate time frame.

An appropriate time frame will be a function of the useful lifespan of the data being protected. For example, an instruction to buy a certain stock may only need to be kept secret for a few minutes whereas state secrets may need to remain confidential indefinitely.

An encryption scheme can be broken by trying all possible keys to see which one the communicating parties are using (assuming that the class of encryption functions is public knowledge). This is called an *exhaustive search* of the key space. It follows then that the number of keys (i.e., the size of the key space) should be large enough to make this approach computationally infeasible. It is the objective of a designer of an encryption scheme that this be the best approach to break the system.

Frequently cited in the literature are *Kerckhoffs' desiderata*, a set of requirements for cipher systems. They are given here essentially as Kerckhoffs originally stated them:

1. the system should be, if not theoretically unbreakable, unbreakable in practice;
2. compromise of the system details should not inconvenience the correspondents;
3. the key should be rememberable without notes and easily changed;
4. the cryptogram should be transmissible by telegraph;
5. the encryption apparatus should be portable and operable by a single person; and
6. the system should be easy, requiring neither the knowledge of a long list of rules nor mental strain.

This list of requirements was articulated in 1883 and, for the most part, remains useful today. Point 2 allows that the class of encryption transformations being used be publicly known and that the security of the system should reside only in the key chosen.

Information security in general

So far the terminology has been restricted to encryption and decryption with the goal of privacy in mind. Information security is much broader, encompassing such things as authentication and data integrity. A few more general definitions, pertinent to discussions later in the book, are given next.

- An *information security service* is a method to provide some specific aspect of security. For example, integrity of transmitted data is a security objective, and a method to ensure this aspect is an information security service.

- *Breaking* an information security service (which often involves more than simply encryption) implies defeating the objective of the intended service.
- A *passive adversary* is an adversary who is capable only of reading information from an unsecured channel.
- An *active adversary* is an adversary who may also transmit, alter, or delete information on an unsecured channel.

Cryptology

- *Cryptanalysis* is the study of mathematical techniques for attempting to defeat cryptographic techniques, and, more generally, information security services.
- A *cryptanalyst* is someone who engages in cryptanalysis.
- *Cryptology* is the study of cryptography (Definition 1.1) and cryptanalysis.
- A *cryptosystem* is a general term referring to a set of cryptographic primitives used to provide information security services. Most often the term is used in conjunction with primitives providing confidentiality, i.e., encryption.

Cryptographic techniques are typically divided into two generic types: *symmetric-key* and *public-key*. Encryption methods of these types will be discussed separately in §1.5 and §1.8. Other definitions and terminology will be introduced as required.

1.5 Symmetric-key encryption

§1.5 considers symmetric-key encryption. Public-key encryption is the topic of §1.8.

1.5.1 Overview of block ciphers and stream ciphers

1.24 Definition Consider an encryption scheme consisting of the sets of encryption and decryption transformations $\{E_e: e \in \mathcal{K}\}$ and $\{D_d: d \in \mathcal{K}\}$, respectively, where $\mathcal{K}$ is the key space. The encryption scheme is said to be *symmetric-key* if for each associated encryption/decryption key pair (e, d) , it is computationally “easy” to determine d knowing only e , and to determine e from d .

Since $e = d$ in most practical symmetric-key encryption schemes, the term symmetric-key becomes appropriate. Other terms used in the literature are *single-key*, *one-key*, *private-key*,² and *conventional* encryption. Example 1.25 illustrates the idea of symmetric-key encryption.

1.25 Example (*symmetric-key encryption*) Let $\mathcal{A} = \{A, B, C, \dots, X, Y, Z\}$ be the English alphabet. Let $\mathcal{M}$ and $\mathcal{C}$ be the set of all strings of length five over $\mathcal{A}$. The key e is chosen to be a permutation on $\mathcal{A}$. To encrypt, an English message is broken up into groups each having five letters (with appropriate padding if the length of the message is not a multiple of five) and a permutation e is applied to each letter one at a time. To decrypt, the inverse permutation $d = e^{-1}$ is applied to each letter of the ciphertext. For instance, suppose that the key e is chosen to be the permutation which maps each letter to the one which is three positions to its right, as shown below

$$e = \begin{pmatrix} A & B & C & D & E & F & G & H & I & J & K & L & M & N & O & P & Q & R & S & T & U & V & W & X & Y & Z \\ D & E & F & G & H & I & J & K & L & M & N & O & P & Q & R & S & T & U & V & W & X & Y & Z & A & B & C \end{pmatrix}$$

²Private key is a term also used in quite a different context (see §1.8). The term will be reserved for the latter usage in this book.

Handbook of Applied Cryptography by A. Menezes, P. van Oorschot and S. Vanstone.

A message

$m = \text{THISC IPHER IS CER TAINL YNOTS ECURE}$

is encrypted to

$c = E_e(m) = \text{WKLVF LSKHU LVFHU WDLQO BQRWV HFXUH.}$ $\square$

A two-party communication using symmetric-key encryption can be described by the block diagram of Figure 1.7, which is Figure 1.6 with the addition of the secure (both con-

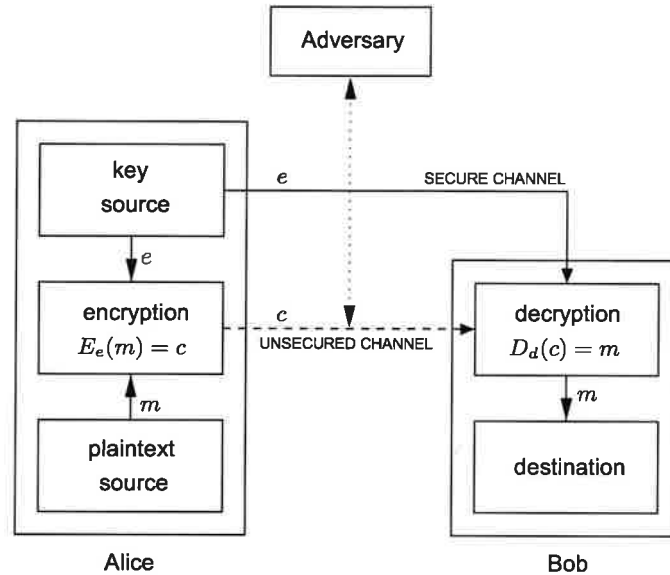


Figure 1.7: Two-party communication using encryption, with a secure channel for key exchange. The decryption key d can be efficiently computed from the encryption key e .

fidential and authentic) channel. One of the major issues with symmetric-key systems is to find an efficient method to agree upon and exchange keys securely. This problem is referred to as the *key distribution problem* (see Chapters 12 and 13).

It is assumed that all parties know the set of encryption/decryption transformations (i.e., they all know the encryption scheme). As has been emphasized several times the only information which should be required to be kept secret is the key d . However, in symmetric-key encryption, this means that the key e must also be kept secret, as d can be deduced from e . In Figure 1.7 the encryption key e is transported from one entity to the other with the understanding that both can construct the decryption key d .

There are two classes of symmetric-key encryption schemes which are commonly distinguished: *block ciphers* and *stream ciphers*.

1.26 Definition A *block cipher* is an encryption scheme which breaks up the plaintext messages to be transmitted into strings (called *blocks*) of a fixed length t over an alphabet $\mathcal{A}$, and encrypts one block at a time.

Most well-known symmetric-key encryption techniques are block ciphers. A number of examples of these are given in Chapter 7. Two important classes of block ciphers are *substitution ciphers* and *transposition ciphers* (§1.5.2). Product ciphers (§1.5.3) combine

©1997 by CRC Press, Inc. — See accompanying notice at front of chapter.

these. Stream ciphers are considered in §1.5.4, while comments on the key space follow in §1.5.5.

1.5.2 Substitution ciphers and transposition ciphers

Substitution ciphers are block ciphers which replace symbols (or groups of symbols) by other symbols or groups of symbols.

Simple substitution ciphers

1.27 Definition Let $\mathcal{A}$ be an alphabet of q symbols and $\mathcal{M}$ be the set of all strings of length t over $\mathcal{A}$. Let $\mathcal{K}$ be the set of all permutations on the set $\mathcal{A}$. Define for each $e \in \mathcal{K}$ an encryption transformation E_e as:

$$E_e(m) = (e(m_1)e(m_2) \cdots e(m_t)) = (c_1c_2 \cdots c_t) = c,$$

where $m = (m_1m_2 \cdots m_t) \in \mathcal{M}$. In other words, for each symbol in a t -tuple, replace (substitute) it by another symbol from $\mathcal{A}$ according to some fixed permutation e . To decrypt $c = (c_1c_2 \cdots c_t)$ compute the inverse permutation $d = e^{-1}$ and

$$D_d(c) = (d(c_1)d(c_2) \cdots d(c_t)) = (m_1m_2 \cdots m_t) = m.$$

E_e is called a *simple substitution cipher* or a *mono-alphabetic substitution cipher*.

The number of distinct substitution ciphers is $q!$ and is independent of the block size in the cipher. Example 1.25 is an example of a simple substitution cipher of block length five.

Simple substitution ciphers over small block sizes provide inadequate security even when the key space is extremely large. If the alphabet is the English alphabet as in Example 1.25, then the size of the key space is $26! \approx 4 \times 10^{26}$, yet the key being used can be determined quite easily by examining a modest amount of ciphertext. This follows from the simple observation that the distribution of letter frequencies is preserved in the ciphertext. For example, the letter E occurs more frequently than the other letters in ordinary English text. Hence the letter occurring most frequently in a sequence of ciphertext blocks is most likely to correspond to the letter E in the plaintext. By observing a modest quantity of ciphertext blocks, a cryptanalyst can determine the key.

Homophonic substitution ciphers

1.28 Definition To each symbol $a \in \mathcal{A}$, associate a set $H(a)$ of strings of t symbols, with the restriction that the sets $H(a)$, $a \in \mathcal{A}$, be pairwise disjoint. A *homophonic substitution cipher* replaces each symbol a in a plaintext message block with a randomly chosen string from $H(a)$. To decrypt a string c of t symbols, one must determine an $a \in \mathcal{A}$ such that $c \in H(a)$. The key for the cipher consists of the sets $H(a)$.

1.29 Example (homophonic substitution cipher) Consider $\mathcal{A} = \{a, b\}$, $H(a) = \{00, 10\}$, and $H(b) = \{01, 11\}$. The plaintext message block ab encrypts to one of the following: 0001, 0011, 1001, 1011. Observe that the codomain of the encryption function (for messages of length two) consists of the following pairwise disjoint sets of four-element bitstrings:

$$\begin{aligned} aa &\longrightarrow \{0000, 0010, 1000, 1010\} \\ ab &\longrightarrow \{0001, 0011, 1001, 1011\} \\ ba &\longrightarrow \{0100, 0110, 1100, 1110\} \\ bb &\longrightarrow \{0101, 0111, 1101, 1111\} \end{aligned}$$

Any 4-bitstring uniquely identifies a codomain element, and hence a plaintext message. $\square$

Handbook of Applied Cryptography by A. Menezes, P. van Oorschot and S. Vanstone.

Often the symbols do not occur with equal frequency in plaintext messages. With a simple substitution cipher this non-uniform frequency property is reflected in the ciphertext as illustrated in Example 1.25. A homophonic cipher can be used to make the frequency of occurrence of ciphertext symbols more uniform, at the expense of data expansion. Decryption is not as easily performed as it is for simple substitution ciphers.

Polyalphabetic substitution ciphers

1.30 Definition A *polyalphabetic substitution cipher* is a block cipher with block length t over an alphabet $\mathcal{A}$ having the following properties:

- (i) the key space $\mathcal{K}$ consists of all ordered sets of t permutations $(p_1, p_2, \dots, p_t)$, where each permutation p_i is defined on the set $\mathcal{A}$;
- (ii) encryption of the message $m = (m_1 m_2 \dots m_t)$ under the key $e = (p_1, p_2, \dots, p_t)$ is given by $E_e(m) = (p_1(m_1) p_2(m_2) \dots p_t(m_t))$; and
- (iii) the decryption key associated with $e = (p_1, p_2, \dots, p_t)$ is $d = (p_1^{-1}, p_2^{-1}, \dots, p_t^{-1})$.

1.31 Example (*Vigenère cipher*) Let $\mathcal{A} = \{A, B, C, \dots, X, Y, Z\}$ and $t = 3$. Choose $e = (p_1, p_2, p_3)$, where p_1 maps each letter to the letter three positions to its right in the alphabet, p_2 to the one seven positions to its right, and p_3 ten positions to its right. If

$m = \text{THI SCI PHE RIS CER TAI NLY NOT SEC URE}$

then

$$c = E_e(m) = \text{WOS VJS SOO UPC FLB WHS QSI QVD VLM XYO}. \quad \square$$

Polyalphabetic ciphers have the advantage over simple substitution ciphers that symbol frequencies are not preserved. In the example above, the letter E is encrypted to both O and L. However, polyalphabetic ciphers are not significantly more difficult to cryptanalyze, the approach being similar to the simple substitution cipher. In fact, once the block length t is determined, the ciphertext letters can be divided into t groups (where group i , $1 \leq i \leq t$, consists of those ciphertext letters derived using permutation p_i), and a frequency analysis can be done on each group.

Transposition ciphers

Another class of symmetric-key ciphers is the simple transposition cipher, which simply permutes the symbols in a block.

1.32 Definition Consider a symmetric-key block encryption scheme with block length t . Let $\mathcal{K}$ be the set of all permutations on the set $\{1, 2, \dots, t\}$. For each $e \in \mathcal{K}$ define the encryption function

$$E_e(m) = (m_{e(1)} m_{e(2)} \dots m_{e(t)})$$

where $m = (m_1 m_2 \dots m_t) \in \mathcal{M}$, the message space. The set of all such transformations is called a *simple transposition cipher*. The decryption key corresponding to e is the inverse permutation $d = e^{-1}$. To decrypt $c = (c_1 c_2 \dots c_t)$, compute $D_d(c) = (c_{d(1)} c_{d(2)} \dots c_{d(t)})$.

A simple transposition cipher preserves the number of symbols of a given type within a block, and thus is easily cryptanalyzed.

1.5.3 Composition of ciphers

In order to describe product ciphers, the concept of composition of functions is introduced. Compositions are a convenient way of constructing more complicated functions from simpler ones.

Composition of functions

1.33 Definition Let $\mathcal{S}$, $\mathcal{T}$, and $\mathcal{U}$ be finite sets and let $f: \mathcal{S} \rightarrow \mathcal{T}$ and $g: \mathcal{T} \rightarrow \mathcal{U}$ be functions. The *composition* of g with f , denoted $g \circ f$ (or simply gf), is a function from $\mathcal{S}$ to $\mathcal{U}$ as illustrated in Figure 1.8 and defined by $(g \circ f)(x) = g(f(x))$ for all $x \in \mathcal{S}$.

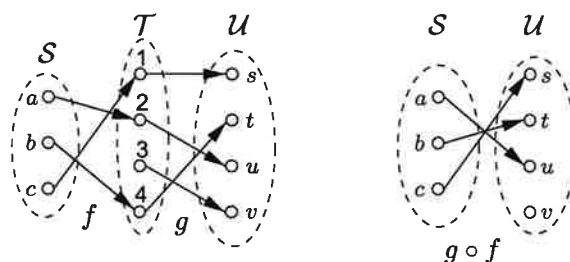


Figure 1.8: The composition $g \circ f$ of functions g and f .

Composition can be easily extended to more than two functions. For functions $f_1, f_2, \dots, f_t$, one can define $f_t \circ \dots \circ f_2 \circ f_1$, provided that the domain of f_t equals the codomain of f_{t-1} and so on.

Compositions and involutions

Involutions were introduced in §1.3.3 as a simple class of functions with an interesting property: $E_k(E_k(x)) = x$ for all x in the domain of E_k ; that is, $E_k \circ E_k$ is the identity function.

1.34 Remark (composition of involutions) The composition of two involutions is not necessarily an involution, as illustrated in Figure 1.9. However, involutions may be composed to get somewhat more complicated functions whose inverses are easy to find. This is an important feature for decryption. For example if $E_{k_1}, E_{k_2}, \dots, E_{k_t}$ are involutions then the inverse of $E_k = E_{k_1} E_{k_2} \dots E_{k_t}$ is $E_k^{-1} = E_{k_t} E_{k_{t-1}} \dots E_{k_1}$, the composition of the involutions in the reverse order.

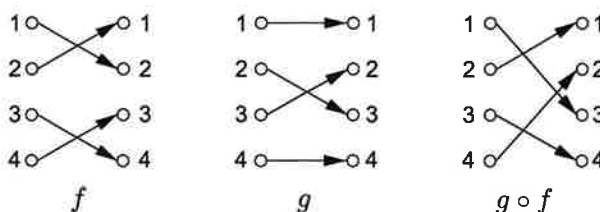


Figure 1.9: The composition $g \circ f$ of involutions g and f is not an involution.

Product ciphers

Simple substitution and transposition ciphers individually do not provide a very high level of security. However, by combining these transformations it is possible to obtain strong ciphers. As will be seen in Chapter 7 some of the most practical and effective symmetric-key systems are product ciphers. One example of a *product cipher* is a composition of $t \geq 2$ transformations $E_{k_1} E_{k_2} \cdots E_{k_t}$ where each E_{k_i} , $1 \leq i \leq t$, is either a substitution or a transposition cipher. For the purpose of this introduction, let the composition of a substitution and a transposition be called a *round*.

- 1.35 Example (product cipher)** Let $\mathcal{M} = \mathcal{C} = \mathcal{K}$ be the set of all binary strings of length six. The number of elements in $\mathcal{M}$ is $2^6 = 64$. Let $m = (m_1 m_2 \cdots m_6)$ and define

$$\begin{aligned} E_k^{(1)}(m) &= m \oplus k, \text{ where } k \in \mathcal{K}, \\ E^{(2)}(m) &= (m_4 m_5 m_6 m_1 m_2 m_3). \end{aligned}$$

Here, $\oplus$ is the *exclusive-OR* (XOR) operation defined as follows: $0 \oplus 0 = 0$, $0 \oplus 1 = 1$, $1 \oplus 0 = 1$, $1 \oplus 1 = 0$. $E_k^{(1)}$ is a polyalphabetic substitution cipher and $E^{(2)}$ is a transposition cipher (not involving the key). The product $E_k^{(1)} E^{(2)}$ is a round. While here the transposition cipher is very simple and is not determined by the key, this need not be the case. $\square$

- 1.36 Remark (confusion and diffusion)** A substitution in a round is said to add *confusion* to the encryption process whereas a transposition is said to add *diffusion*. Confusion is intended to make the relationship between the key and ciphertext as complex as possible. Diffusion refers to rearranging or spreading out the bits in the message so that any redundancy in the plaintext is spread out over the ciphertext. A round then can be said to add both confusion and diffusion to the encryption. Most modern block cipher systems apply a number of rounds in succession to encrypt plaintext.

1.5.4 Stream ciphers

Stream ciphers form an important class of symmetric-key encryption schemes. They are, in one sense, very simple block ciphers having block length equal to one. What makes them useful is the fact that the encryption transformation can change for each symbol of plaintext being encrypted. In situations where transmission errors are highly probable, stream ciphers are advantageous because they have no error propagation. They can also be used when the data must be processed one symbol at a time (e.g., if the equipment has no memory or buffering of data is limited).

- 1.37 Definition** Let $\mathcal{K}$ be the key space for a set of encryption transformations. A sequence of symbols $e_1 e_2 e_3 \cdots e_i \in \mathcal{K}$, is called a *keystream*.
- 1.38 Definition** Let $\mathcal{A}$ be an alphabet of q symbols and let E_e be a simple substitution cipher with block length 1 where $e \in \mathcal{K}$. Let $m_1 m_2 m_3 \cdots$ be a plaintext string and let $e_1 e_2 e_3 \cdots$ be a keystream from $\mathcal{K}$. A *stream cipher* takes the plaintext string and produces a ciphertext string $c_1 c_2 c_3 \cdots$ where $c_i = E_{e_i}(m_i)$. If d_i denotes the inverse of e_i , then $D_{d_i}(c_i) = m_i$ decrypts the ciphertext string.

A stream cipher applies simple encryption transformations according to the keystream being used. The keystream could be generated at random, or by an algorithm which generates the keystream from an initial small keystream (called a *seed*), or from a seed and previous ciphertext symbols. Such an algorithm is called a *keystream generator*.

The Vernam cipher

A motivating factor for the Vernam cipher was its simplicity and ease of implementation.

1.39 Definition The *Vernam Cipher* is a stream cipher defined on the alphabet $\mathcal{A} = \{0, 1\}$. A binary message $m_1 m_2 \dots m_t$ is operated on by a binary key string $k_1 k_2 \dots k_t$ of the same length to produce a ciphertext string $c_1 c_2 \dots c_t$ where

$$c_i = m_i \oplus k_i, \quad 1 \leq i \leq t.$$

If the key string is randomly chosen and never used again, the Vernam cipher is called a *one-time system* or a *one-time pad*.

To see how the Vernam cipher corresponds to Definition 1.38, observe that there are precisely two substitution ciphers on the set $\mathcal{A}$. One is simply the identity map E_0 which sends 0 to 0 and 1 to 1; the other E_1 sends 0 to 1 and 1 to 0. When the keystream contains a 0, apply E_0 to the corresponding plaintext symbol; otherwise, apply E_1 .

If the key string is reused there are ways to attack the system. For example, if $c_1 c_2 \dots c_t$ and $c'_1 c'_2 \dots c'_t$ are two ciphertext strings produced by the same keystream $k_1 k_2 \dots k_t$ then

$$c_i = m_i \oplus k_i, \quad c'_i = m'_i \oplus k_i$$

and $c_i \oplus c'_i = m_i \oplus m'_i$. The redundancy in the latter may permit cryptanalysis.

The one-time pad can be shown to be theoretically unbreakable. That is, if a cryptanalyst has a ciphertext string $c_1 c_2 \dots c_t$ encrypted using a random key string which has been used only once, the cryptanalyst can do no better than guess at the plaintext being any binary string of length t (i.e., t -bit binary strings are equally likely as plaintext). It has been proven that to realize an unbreakable system requires a random key of the same length as the message. This reduces the practicality of the system in all but a few specialized situations. Reportedly until very recently the communication line between Moscow and Washington was secured by a one-time pad. Transport of the key was done by trusted courier.

1.5.5 The key space

The size of the key space is the number of encryption/decryption key pairs that are available in the cipher system. A key is typically a compact way to specify the encryption transformation (from the set of all encryption transformations) to be used. For example, a transposition cipher of block length t has $t!$ encryption functions from which to select. Each can be simply described by a permutation which is called the key.

It is a great temptation to relate the security of the encryption scheme to the size of the key space. The following statement is important to remember.

1.40 Fact A necessary, but usually not sufficient, condition for an encryption scheme to be secure is that the key space be large enough to preclude exhaustive search.

For instance, the simple substitution cipher in Example 1.25 has a key space of size $26! \approx 4 \times 10^{26}$. The polyalphabetic substitution cipher of Example 1.31 has a key space of size $(26!)^3 \approx 7 \times 10^{79}$. Exhaustive search of either key space is completely infeasible, yet both ciphers are relatively weak and provide little security.

Handbook of Applied Cryptography by A. Menezes, P. van Oorschot and S. Vanstone.

1.6 Digital signatures

A cryptographic primitive which is fundamental in authentication, authorization, and non-repudiation is the *digital signature*. The purpose of a digital signature is to provide a means for an entity to bind its identity to a piece of information. The process of *signing* entails transforming the message and some secret information held by the entity into a tag called a *signature*. A generic description follows.

Nomenclature and set-up

- $\mathcal{M}$ is the set of messages which can be signed.
- $\mathcal{S}$ is a set of elements called *signatures*, possibly binary strings of a fixed length.
- S_A is a transformation from the message set $\mathcal{M}$ to the signature set $\mathcal{S}$, and is called a *signing transformation* for entity A .³ The transformation S_A is kept secret by A , and will be used to create signatures for messages from $\mathcal{M}$.
- V_A is a transformation from the set $\mathcal{M} \times \mathcal{S}$ to the set $\{\text{true}, \text{false}\}$.⁴ V_A is called a *verification transformation* for A 's signatures, is publicly known, and is used by other entities to verify signatures created by A .

1.41 Definition The transformations S_A and V_A provide a *digital signature scheme* for A . Occasionally the term *digital signature mechanism* is used.

1.42 Example (digital signature scheme) $\mathcal{M} = \{m_1, m_2, m_3\}$ and $\mathcal{S} = \{s_1, s_2, s_3\}$. The left side of Figure 1.10 displays a signing function S_A from the set $\mathcal{M}$ and, the right side, the corresponding verification function V_A . $\square$

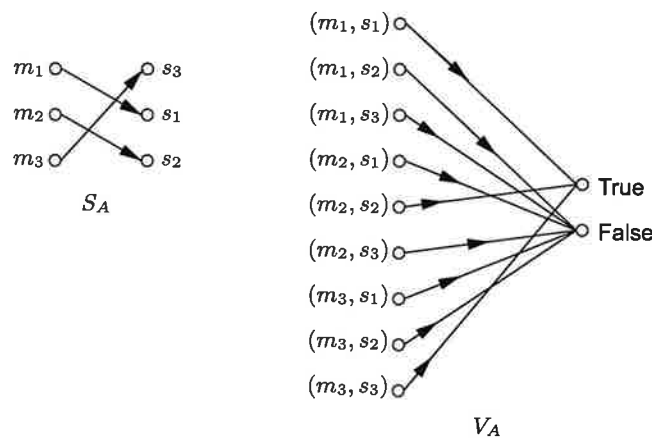


Figure 1.10: A signing and verification function for a digital signature scheme.

³The names of Alice and Bob are usually abbreviated to A and B , respectively.

⁴ $\mathcal{M} \times \mathcal{S}$ consists of all pairs (m, s) where $m \in \mathcal{M}$, $s \in \mathcal{S}$, called the *Cartesian product* of $\mathcal{M}$ and $\mathcal{S}$.

Signing procedure

Entity A (the *signer*) creates a signature for a message $m \in \mathcal{M}$ by doing the following:

1. Compute $s = S_A(m)$.
2. Transmit the pair (m, s) . s is called the *signature* for message m .

Verification procedure

To verify that a signature s on a message m was created by A , an entity B (the *verifier*) performs the following steps:

1. Obtain the verification function V_A of A .
2. Compute $u = V_A(m, s)$.
3. Accept the signature as having been created by A if $u = \text{true}$, and reject the signature if $u = \text{false}$.

1.43 Remark (concise representation) The transformations S_A and V_A are typically characterized more compactly by a key; that is, there is a class of signing and verification algorithms publicly known, and each algorithm is identified by a key. Thus the signing algorithm S_A of A is determined by a key k_A and A is only required to keep k_A secret. Similarly, the verification algorithm V_A of A is determined by a key l_A which is made public.

1.44 Remark (handwritten signatures) Handwritten signatures could be interpreted as a special class of digital signatures. To see this, take the set of signatures $\mathcal{S}$ to contain only one element which is the handwritten signature of A , denoted by s_A . The verification function simply checks if the signature on a message purportedly signed by A is s_A .

An undesirable feature in Remark 1.44 is that the signature is not message-dependent. Hence, further constraints are imposed on digital signature mechanisms as next discussed.

Properties required for signing and verification functions

There are several properties which the signing and verification transformations must satisfy.

- (a) s is a valid signature of A on message m if and only if $V_A(m, s) = \text{true}$.
- (b) It is computationally infeasible for any entity other than A to find, for any $m \in \mathcal{M}$, an $s \in \mathcal{S}$ such that $V_A(m, s) = \text{true}$.

Figure 1.10 graphically displays property (a). There is an arrowed line in the diagram for V_A from (m_i, s_j) to true provided there is an arrowed line from m_i to s_j in the diagram for S_A . Property (b) provides the security for the method – the signature uniquely binds A to the message which is signed.

No one has yet formally proved that digital signature schemes satisfying (b) exist (although existence is widely believed to be true); however, there are some very good candidates. §1.8.3 introduces a particular class of digital signatures which arise from public-key encryption techniques. Chapter 11 describes a number of digital signature mechanisms which are believed to satisfy the two properties cited above. Although the description of a digital signature given in this section is quite general, it can be broadened further, as presented in §11.2.

1.7 Authentication and identification

Authentication is a term which is used (and often abused) in a very broad sense. By itself it has little meaning other than to convey the idea that some means has been provided to guarantee that entities are who they claim to be, or that information has not been manipulated by unauthorized parties. Authentication is specific to the security objective which one is trying to achieve. Examples of specific objectives include access control, entity authentication, message authentication, data integrity, non-repudiation, and key authentication. These instances of authentication are dealt with at length in Chapters 9 through 13. For the purposes of this chapter, it suffices to give a brief introduction to authentication by describing several of the most obvious applications.

Authentication is one of the most important of all information security objectives. Until the mid 1970s it was generally believed that secrecy and authentication were intrinsically connected. With the discovery of hash functions (§1.9) and digital signatures (§1.6), it was realized that secrecy and authentication were truly separate and independent information security objectives. It may at first not seem important to separate the two but there are situations where it is not only useful but essential. For example, if a two-party communication between Alice and Bob is to take place where Alice is in one country and Bob in another, the host countries might not permit secrecy on the channel; one or both countries might want the ability to monitor all communications. Alice and Bob, however, would like to be assured of the identity of each other, and of the integrity and origin of the information they send and receive.

The preceding scenario illustrates several independent aspects of authentication. If Alice and Bob desire assurance of each other's identity, there are two possibilities to consider.

1. Alice and Bob could be communicating with no appreciable time delay. That is, they are both active in the communication in "real time".
2. Alice or Bob could be exchanging messages with some delay. That is, messages might be routed through various networks, stored, and forwarded at some later time.

In the first instance Alice and Bob would want to verify identities in real time. This might be accomplished by Alice sending Bob some challenge, to which Bob is the only entity which can respond correctly. Bob could perform a similar action to identify Alice. This type of authentication is commonly referred to as *entity authentication* or more simply *identification*.

For the second possibility, it is not convenient to challenge and await response, and moreover the communication path may be only in one direction. Different techniques are now required to authenticate the originator of the message. This form of authentication is called *data origin authentication*.

1.7.1 Identification

1.45 Definition An *identification* or *entity authentication* technique assures one party (through acquisition of corroborative evidence) of both the identity of a second party involved, and that the second was active at the time the evidence was created or acquired.

Typically the only data transmitted is that necessary to identify the communicating parties. The entities are both active in the communication, giving a timeliness guarantee.

1.46 Example (identification) *A* calls *B* on the telephone. If *A* and *B* know each other then entity authentication is provided through voice recognition. Although not foolproof, this works effectively in practice. $\square$

1.47 Example (identification) Person *A* provides to a banking machine a personal identification number (PIN) along with a magnetic stripe card containing information about *A*. The banking machine uses the information on the card and the PIN to verify the identity of the card holder. If verification succeeds, *A* is given access to various services offered by the machine. $\square$

Example 1.46 is an instance of *mutual authentication* whereas Example 1.47 only provides *unilateral authentication*. Numerous mechanisms and protocols devised to provide mutual or unilateral authentication are discussed in Chapter 10.

1.7.2 Data origin authentication

1.48 Definition *Data origin authentication* or *message authentication* techniques provide to one party which receives a message assurance (through corroborative evidence) of the identity of the party which originated the message.

Often a message is provided to *B* along with additional information so that *B* can determine the identity of the entity who originated the message. This form of authentication typically provides no guarantee of timeliness, but is useful in situations where one of the parties is not active in the communication.

1.49 Example (need for data origin authentication) *A* sends to *B* an electronic mail message (e-mail). The message may travel through various network communications systems and be stored for *B* to retrieve at some later time. *A* and *B* are usually not in direct communication. *B* would like some means to verify that the message received and purportedly created by *A* did indeed originate from *A*. $\square$

Data origin authentication implicitly provides data integrity since, if the message was modified during transmission, *A* would no longer be the originator.

1.8 Public-key cryptography

The concept of public-key encryption is simple and elegant, but has far-reaching consequences.

1.8.1 Public-key encryption

Let $\{E_e : e \in \mathcal{K}\}$ be a set of encryption transformations, and let $\{D_d : d \in \mathcal{K}\}$ be the set of corresponding decryption transformations, where $\mathcal{K}$ is the key space. Consider any pair of associated encryption/decryption transformations (E_e, D_d) and suppose that each pair has the property that knowing E_e it is computationally infeasible, given a random ciphertext $c \in \mathcal{C}$, to find the message $m \in \mathcal{M}$ such that $E_e(m) = c$. This property implies that given e it is infeasible to determine the corresponding decryption key d . (Of course e and d are

Handbook of Applied Cryptography by A. Menezes, P. van Oorschot and S. Vanstone.

simply means to describe the encryption and decryption functions, respectively.) E_e is being viewed here as a trapdoor one-way function (Definition 1.16) with d being the trapdoor information necessary to compute the inverse function and hence allow decryption. This is unlike symmetric-key ciphers where e and d are essentially the same.

Under these assumptions, consider the two-party communication between Alice and Bob illustrated in Figure 1.11. Bob selects the key pair (e, d) . Bob sends the encryption key e (called the *public key*) to Alice over any channel but keeps the decryption key d (called the *private key*) secure and secret. Alice may subsequently send a message m to Bob by applying the encryption transformation determined by Bob's public key to get $c = E_e(m)$. Bob decrypts the ciphertext c by applying the inverse transformation D_d uniquely determined by d .

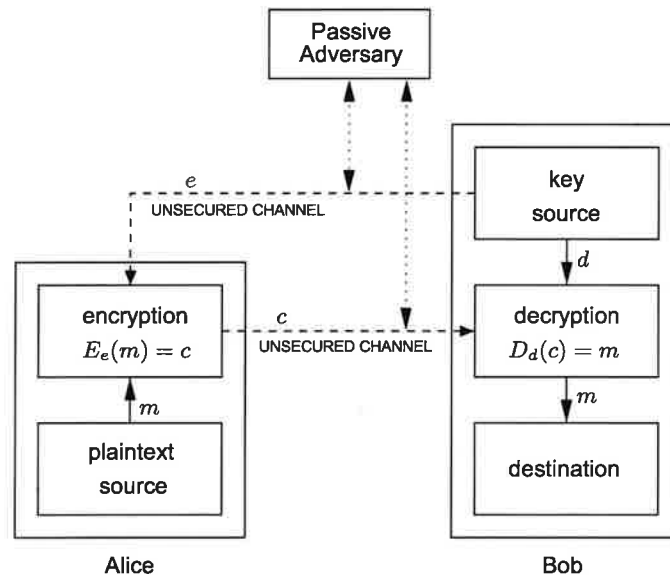


Figure 1.11: Encryption using public-key techniques.

Notice how Figure 1.11 differs from Figure 1.7 for a symmetric-key cipher. Here the encryption key is transmitted to Alice over an unsecured channel. This unsecured channel may be the same channel on which the ciphertext is being transmitted (but see §1.8.2).

Since the encryption key e need not be kept secret, it may be made public. Any entity can subsequently send encrypted messages to Bob which only Bob can decrypt. Figure 1.12 illustrates this idea, where A_1 , A_2 , and A_3 are distinct entities. Note that if A_1 destroys message m_1 after encrypting it to c_1 , then even A_1 cannot recover m_1 from c_1 .

As a physical analogue, consider a metal box with the lid secured by a combination lock. The combination is known only to Bob. If the lock is left open and made publicly available then anyone can place a message inside and lock the lid. Only Bob can retrieve the message. Even the entity which placed the message into the box is unable to retrieve it.

Public-key encryption, as described here, assumes that knowledge of the public key e does not allow computation of the private key d . In other words, this assumes the existence of trapdoor one-way functions (§1.3.1(iii)).

1.50 Definition Consider an encryption scheme consisting of the sets of encryption and decryp-

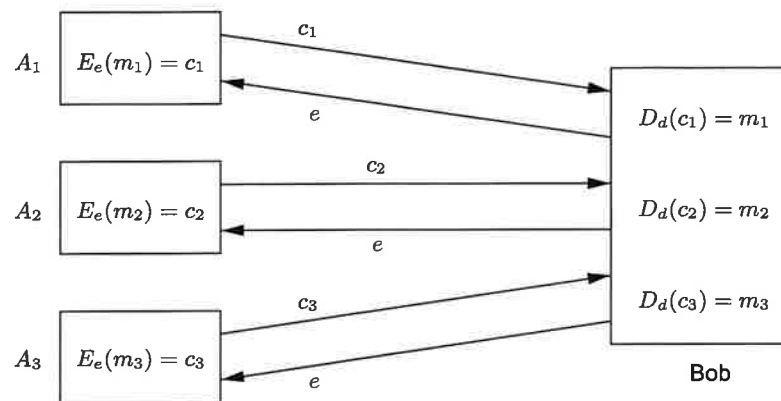


Figure 1.12: Schematic use of public-key encryption.

tion transformations $\{E_e : e \in \mathcal{K}\}$ and $\{D_d : d \in \mathcal{K}\}$, respectively. The encryption method is said to be a *public-key encryption scheme* if for each associated encryption/decryption pair (e, d) , one key e (the *public key*) is made publicly available, while the other d (the *private key*) is kept secret. For the scheme to be *secure*, it must be computationally infeasible to compute d from e .

1.51 Remark (private key vs. secret key) To avoid ambiguity, a common convention is to use the term *private key* in association with public-key cryptosystems, and *secret key* in association with symmetric-key cryptosystems. This may be motivated by the following line of thought: it takes two or more parties to *share* a secret, but a key is truly *private* only when one party alone knows it.

There are many schemes known which are widely believed to be secure public-key encryption methods, but none have been mathematically proven to be secure independent of qualifying assumptions. This is not unlike the symmetric-key case where the only system which has been proven secure is the one-time pad (§1.5.4).

1.8.2 The necessity of authentication in public-key systems

It would appear that public-key cryptography is an ideal system, not requiring a secure channel to pass the encryption key. This would imply that two entities could communicate over an unsecured channel without ever having met to exchange keys. Unfortunately, this is not the case. Figure 1.13 illustrates how an active adversary can defeat the system (decrypt messages intended for a second entity) without breaking the encryption system. This is a type of *impersonation* and is an example of *protocol failure* (see §1.10). In this scenario the adversary impersonates entity B by sending entity A a public key e' which A assumes (incorrectly) to be the public key of B . The adversary intercepts encrypted messages from A to B , decrypts with its own private key d' , re-encrypts the message under B 's public key e , and sends it on to B . This highlights the necessity to *authenticate* public keys to achieve data origin authentication of the public keys themselves. A must be convinced that she is

Handbook of Applied Cryptography by A. Menezes, P. van Oorschot and S. Vanstone.

encrypting under the legitimate public key of B . Fortunately, public-key techniques also allow an elegant solution to this problem (see §1.11).

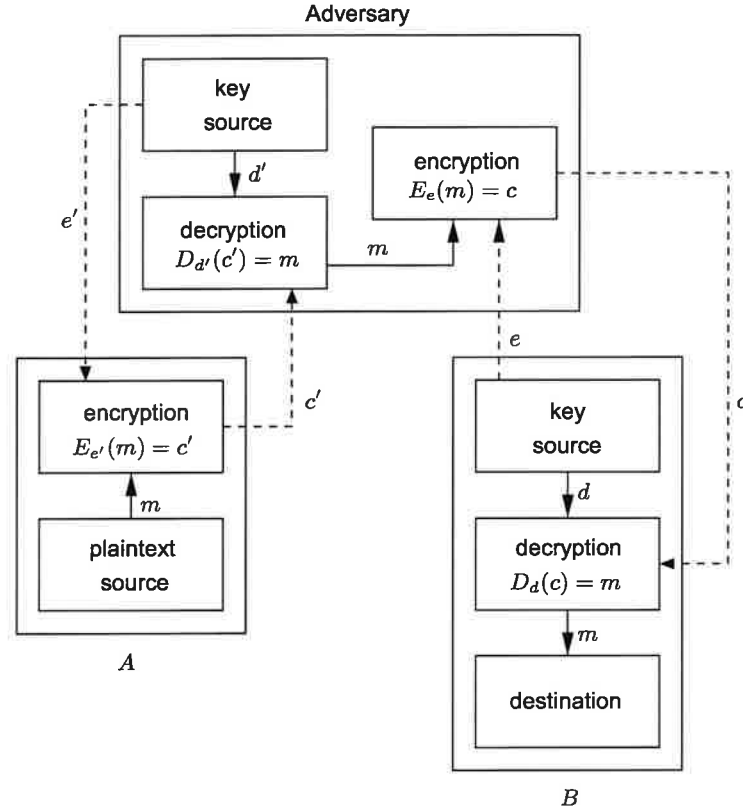


Figure 1.13: An impersonation attack on a two-party communication.

1.8.3 Digital signatures from reversible public-key encryption

This section considers a class of digital signature schemes which is based on public-key encryption systems of a particular type.

Suppose E_e is a public-key encryption transformation with message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$. Suppose further that $\mathcal{M} = \mathcal{C}$. If D_d is the decryption transformation corresponding to E_e then since E_e and D_d are both permutations, one has

$$D_d(E_e(m)) = E_e(D_d(m)) = m, \text{ for all } m \in \mathcal{M}.$$

A public-key encryption scheme of this type is called *reversible*.⁵ Note that it is essential that $\mathcal{M} = \mathcal{C}$ for this to be a valid equality for all $m \in \mathcal{M}$; otherwise, $D_d(m)$ will be meaningless for $m \notin \mathcal{C}$.

⁵There is a broader class of digital signatures which can be informally described as arising from *irreversible* cryptographic algorithms. These are described in §11.2.

Construction for a digital signature scheme

1. Let $\mathcal{M}$ be the message space for the signature scheme.
2. Let $\mathcal{C} = \mathcal{M}$ be the signature space $\mathcal{S}$.
3. Let (e, d) be a key pair for the public-key encryption scheme.
4. Define the signing function S_A to be D_d . That is, the signature for a message $m \in \mathcal{M}$ is $s = D_d(m)$.
5. Define the verification function V_A by

$$V_A(m, s) = \begin{cases} \text{true}, & \text{if } E_e(s) = m, \\ \text{false}, & \text{otherwise.} \end{cases}$$

The signature scheme can be simplified further if A only signs messages having a special structure, and this structure is publicly known. Let $\mathcal{M}'$ be a subset of $\mathcal{M}$ where elements of $\mathcal{M}'$ have a well-defined special structure, such that $\mathcal{M}'$ contains only a negligible fraction of messages from the set. For example, suppose that $\mathcal{M}$ consists of all binary strings of length $2t$ for some positive integer t . Let $\mathcal{M}'$ be the subset of $\mathcal{M}$ consisting of all strings where the first t bits are replicated in the last t positions (e.g., 101101 would be in $\mathcal{M}'$ for $t = 3$). If A only signs messages within the subset $\mathcal{M}'$, these are easily recognized by a verifier.

Redefine the verification function V_A as

$$V_A(s) = \begin{cases} \text{true}, & \text{if } E_e(s) \in \mathcal{M}', \\ \text{false}, & \text{otherwise.} \end{cases}$$

Under this new scenario A only needs to transmit the signature s since the message $m = E_e(s)$ can be recovered by applying the verification function. Such a scheme is called a *digital signature scheme with message recovery*. Figure 1.14 illustrates how this signature function is used. The feature of selecting messages of special structure is referred to as selecting messages with *redundancy*.

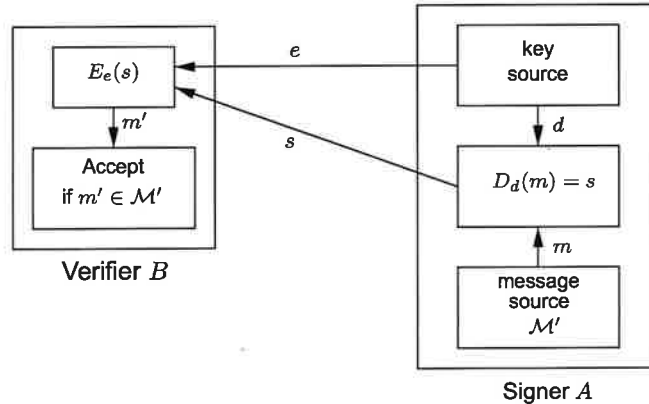


Figure 1.14: A digital signature scheme with message recovery.

The modification presented above is more than a simplification; it is absolutely crucial if one hopes to meet the requirement of property (b) of signing and verification functions (see page 23). To see why this is the case, note that any entity B can select a random element $s \in \mathcal{S}$ as a signature and apply E_e to get $u = E_e(s)$, since $\mathcal{S} = \mathcal{M}$ and E_e is public

knowledge. B may then take the message $m = u$ and the signature on m to be s and transmits (m, s) . It is easy to check that s will verify as a signature created by A for m but in which A has had no part. In this case B has *forged* a signature of A . This is an example of what is called *existential forgery*. (B has produced A 's signature on some message likely not of B 's choosing.)

If $\mathcal{M}'$ contains only a negligible fraction of messages from $\mathcal{M}$, then the probability of some entity forging a signature of A in this manner is negligibly small.

1.52 Remark (*digital signatures vs. confidentiality*) Although digital signature schemes based on reversible public-key encryption are attractive, they require an encryption method as a primitive. There are situations where a digital signature mechanism is required but encryption is forbidden. In such cases these digital signature schemes are inappropriate.

Digital signatures in practice

For digital signatures to be useful in practice, concrete realizations of the preceding concepts should have certain additional properties. A digital signature must

1. be easy to compute by the signer (the signing function should be easy to apply);
2. be easy to verify by anyone (the verification function should be easy to apply); and
3. have an appropriate lifespan, i.e., be computationally secure from forgery until the signature is no longer necessary for its original purpose.

Resolution of disputes

The purpose of a digital signature (or any signature method) is to permit the resolution of disputes. For example, an entity A could at some point deny having signed a message or some other entity B could falsely claim that a signature on a message was produced by A . In order to overcome such problems a *trusted third party* (TTP) or *judge* is required. The TTP must be some entity which all parties involved agree upon in advance.

If A denies that a message m held by B was signed by A , then B should be able to present the signature s_A for m to the TTP along with m . The TTP rules in favor of B if $V_A(m, s_A) = \text{true}$ and in favor of A otherwise. B will accept the decision if B is confident that the TTP has the same verifying transformation V_A as A does. A will accept the decision if A is confident that the TTP used V_A and that S_A has not been compromised. Therefore, fair resolution of disputes requires that the following criteria are met.

Requirements for resolution of disputed signatures

1. S_A and V_A have properties (a) and (b) of page 23.
2. The TTP has an authentic copy of V_A .
3. The signing transformation S_A has been kept secret and remains secure.

These properties are necessary but in practice it might not be possible to guarantee them. For example, the assumption that S_A and V_A have the desired characteristics given in property 1 might turn out to be false for a particular signature scheme. Another possibility is that A claims falsely that S_A was compromised. To overcome these problems requires an agreed method to validate the time period for which A will accept responsibility for the verification transformation. An analogue of this situation can be made with credit card revocation. The holder of a card is responsible until the holder notifies the card issuing company that the card has been lost or stolen. §13.8.2 gives a more indepth discussion of these problems and possible solutions.

1.8.4 Symmetric-key vs. public-key cryptography

Symmetric-key and public-key encryption schemes have various advantages and disadvantages, some of which are common to both. This section highlights a number of these and summarizes features pointed out in previous sections.

(i) Advantages of symmetric-key cryptography

1. Symmetric-key ciphers can be designed to have high rates of data throughput. Some hardware implementations achieve encrypt rates of hundreds of megabytes per second, while software implementations may attain throughput rates in the megabytes per second range.
2. Keys for symmetric-key ciphers are relatively short.
3. Symmetric-key ciphers can be employed as primitives to construct various cryptographic mechanisms including pseudorandom number generators (see Chapter 5), hash functions (see Chapter 9), and computationally efficient digital signature schemes (see Chapter 11), to name just a few.
4. Symmetric-key ciphers can be composed to produce stronger ciphers. Simple transformations which are easy to analyze, but on their own weak, can be used to construct strong product ciphers.
5. Symmetric-key encryption is perceived to have an extensive history, although it must be acknowledged that, notwithstanding the invention of rotor machines earlier, much of the knowledge in this area has been acquired subsequent to the invention of the digital computer, and, in particular, the design of the Data Encryption Standard (see Chapter 7) in the early 1970s.

(ii) Disadvantages of symmetric-key cryptography

1. In a two-party communication, the key must remain secret at both ends.
2. In a large network, there are many key pairs to be managed. Consequently, effective key management requires the use of an unconditionally trusted TTP (Definition 1.65).
3. In a two-party communication between entities A and B , sound cryptographic practice dictates that the key be changed frequently, and perhaps for each communication session.
4. Digital signature mechanisms arising from symmetric-key encryption typically require either large keys for the public verification function or the use of a TTP (see Chapter 11).

(iii) Advantages of public-key cryptography

1. Only the private key must be kept secret (authenticity of public keys must, however, be guaranteed).
2. The administration of keys on a network requires the presence of only a functionally trusted TTP (Definition 1.66) as opposed to an unconditionally trusted TTP. Depending on the mode of usage, the TTP might only be required in an “off-line” manner, as opposed to in real time.
3. Depending on the mode of usage, a private key/public key pair may remain unchanged for considerable periods of time, e.g., many sessions (even several years).
4. Many public-key schemes yield relatively efficient digital signature mechanisms. The key used to describe the public verification function is typically much smaller than for the symmetric-key counterpart.

Handbook of Applied Cryptography by A. Menezes, P. van Oorschot and S. Vanstone.

5. In a large network, the number of keys necessary may be considerably smaller than in the symmetric-key scenario.

(iv) Disadvantages of public-key encryption

1. Throughput rates for the most popular public-key encryption methods are several orders of magnitude slower than the best known symmetric-key schemes.
2. Key sizes are typically much larger than those required for symmetric-key encryption (see Remark 1.53), and the size of public-key signatures is larger than that of tags providing data origin authentication from symmetric-key techniques.
3. No public-key scheme has been proven to be secure (the same can be said for block ciphers). The most effective public-key encryption schemes found to date have their security based on the presumed difficulty of a small set of number-theoretic problems.
4. Public-key cryptography does not have as extensive a history as symmetric-key encryption, being discovered only in the mid 1970s.⁶

Summary of comparison

Symmetric-key and public-key encryption have a number of complementary advantages. Current cryptographic systems exploit the strengths of each. An example will serve to illustrate.

Public-key encryption techniques may be used to establish a key for a symmetric-key system being used by communicating entities *A* and *B*. In this scenario *A* and *B* can take advantage of the long term nature of the public/private keys of the public-key scheme and the performance efficiencies of the symmetric-key scheme. Since data encryption is frequently the most time consuming part of the encryption process, the public-key scheme for key establishment is a small fraction of the total encryption process between *A* and *B*.

To date, the computational performance of public-key encryption is inferior to that of symmetric-key encryption. There is, however, no proof that this must be the case. The important points in practice are:

1. public-key cryptography facilitates efficient signatures (particularly non-repudiation) and key management; and
2. symmetric-key cryptography is efficient for encryption and some data integrity applications.

1.53 Remark (*key sizes: symmetric key vs. private key*) Private keys in public-key systems must be larger (e.g., 1024 bits for RSA) than secret keys in symmetric-key systems (e.g., 64 or 128 bits) because whereas (for secure algorithms) the most efficient attack on symmetric-key systems is an exhaustive key search, all known public-key systems are subject to “short-cut” attacks (e.g., factoring) more efficient than exhaustive search. Consequently, for equivalent security, symmetric keys have bitlengths considerably smaller than that of private keys in public-key systems, e.g., by a factor of 10 or more.

⁶It is, of course, arguable that some public-key schemes which are based on hard mathematical problems have a long history since these problems have been studied for many years. Although this may be true, one must be wary that the mathematics was not studied with this application in mind.

1.9 Hash functions

One of the fundamental primitives in modern cryptography is the cryptographic hash function, often informally called a one-way hash function. A simplified definition for the present discussion follows.

1.54 Definition A *hash function* is a computationally efficient function mapping binary strings of arbitrary length to binary strings of some fixed length, called *hash-values*.

For a hash function which outputs n -bit hash-values (e.g., $n = 128$ or 160) and has desirable properties, the probability that a randomly chosen string gets mapped to a particular n -bit hash-value (image) is 2^{-n} . The basic idea is that a hash-value serves as a compact representative of an input string. To be of cryptographic use, a hash function h is typically chosen such that it is computationally infeasible to find two distinct inputs which hash to a common value (i.e., two *colliding* inputs x and y such that $h(x) = h(y)$), and that given a specific hash-value y , it is computationally infeasible to find an input (pre-image) x such that $h(x) = y$.

The most common cryptographic uses of hash functions are with digital signatures and for data integrity. With digital signatures, a long message is usually hashed (using a publicly available hash function) and only the hash-value is signed. The party receiving the message then hashes the received message, and verifies that the received signature is correct for this hash-value. This saves both time and space compared to signing the message directly, which would typically involve splitting the message into appropriate-sized blocks and signing each block individually. Note here that the inability to find two messages with the same hash-value is a security requirement, since otherwise, the signature on one message hash-value would be the same as that on another, allowing a signer to sign one message and at a later point in time claim to have signed another.

Hash functions may be used for data integrity as follows. The hash-value corresponding to a particular input is computed at some point in time. The integrity of this hash-value is protected in some manner. At a subsequent point in time, to verify that the input data has not been altered, the hash-value is recomputed using the input at hand, and compared for equality with the original hash-value. Specific applications include virus protection and software distribution.

A third application of hash functions is their use in protocols involving a priori commitments, including some digital signature schemes and identification protocols (e.g., see Chapter 10).

Hash functions as discussed above are typically publicly known and involve no secret keys. When used to detect whether the message input has been altered, they are called *modification detection codes* (MDCs). Related to these are hash functions which involve a secret key, and provide data origin authentication (§9.76) as well as data integrity; these are called *message authentication codes* (MACs).

1.10 Protocols and mechanisms

1.55 Definition A *cryptographic protocol (protocol)* is a distributed algorithm defined by a sequence of steps precisely specifying the actions required of two or more entities to achieve a specific security objective.

Handbook of Applied Cryptography by A. Menezes, P. van Oorschot and S. Vanstone.

1.56 Remark (*protocol vs. mechanism*) As opposed to a protocol, a *mechanism* is a more general term encompassing protocols, algorithms (specifying the steps followed by a single entity), and non-cryptographic techniques (e.g., hardware protection and procedural controls) to achieve specific security objectives.

Protocols play a major role in cryptography and are essential in meeting cryptographic goals as discussed in §1.2. Encryption schemes, digital signatures, hash functions, and random number generation are among the primitives which may be utilized to build a protocol.

1.57 Example (*a simple key agreement protocol*) Alice and Bob have chosen a symmetric-key encryption scheme to use in communicating over an unsecured channel. To encrypt information they require a key. The communication protocol is the following:

1. Bob constructs a public-key encryption scheme and sends his public key to Alice over the channel.
2. Alice generates a key for the symmetric-key encryption scheme.
3. Alice encrypts the key using Bob's public key and sends the encrypted key to Bob.
4. Bob decrypts using his private key and recovers the symmetric (secret) key.
5. Alice and Bob begin communicating with privacy by using the symmetric-key system and the common secret key.

This protocol uses basic functions to attempt to realize private communications on an unsecured channel. The basic primitives are the symmetric-key and the public-key encryption schemes. The protocol has shortcomings including the impersonation attack of §1.8.2, but it does convey the idea of a protocol. □

Often the role of public-key encryption in privacy communications is exactly the one suggested by this protocol – public-key encryption is used as a means to exchange keys for subsequent use in symmetric-key encryption, motivated by performance differences between symmetric-key and public-key encryption.

Protocol and mechanism failure

1.58 Definition A *protocol failure* or *mechanism failure* occurs when a mechanism fails to meet the goals for which it was intended, in a manner whereby an adversary gains advantage not by breaking an underlying primitive such as an encryption algorithm directly, but by manipulating the protocol or mechanism itself.

1.59 Example (*mechanism failure*) Alice and Bob are communicating using a stream cipher. Messages which they encrypt are known to have a special form: the first twenty bits carry information which represents a monetary amount. An active adversary can simply XOR an appropriate bitstring into the first twenty bits of ciphertext and change the amount. While the adversary has not been able to read the underlying message, she has been able to alter the transmission. The encryption has not been compromised but the protocol has failed to perform adequately; the inherent assumption that encryption provides data integrity is incorrect. □

1.60 Example (*forward search attack*) Suppose that in an electronic bank transaction the 32-bit field which records the value of the transaction is to be encrypted using a public-key scheme. This simple protocol is intended to provide privacy of the value field – but does it? An adversary could easily take all 2^{32} possible entries that could be plaintext in this field and encrypt them using the public encryption function. (Remember that by the very nature of public-key encryption this function must be available to the adversary.) By comparing

each of the 2^{32} ciphertexts with the one which is actually encrypted in the transaction, the adversary can determine the plaintext. Here the public-key encryption function is not compromised, but rather the way it is used. A closely related attack which applies directly to authentication for access control purposes is the dictionary attack (see §10.2.2). $\square$

1.61 Remark (*causes of protocol failure*) Protocols and mechanisms may fail for a number of reasons, including:

1. weaknesses in a particular cryptographic primitive which may be amplified by the protocol or mechanism;
2. claimed or assumed security guarantees which are overstated or not clearly understood; and
3. the oversight of some principle applicable to a broad class of primitives such as encryption.

Example 1.59 illustrates item 2 if the stream cipher is the one-time pad, and also item 1. Example 1.60 illustrates item 3. See also §1.8.2.

1.62 Remark (*protocol design*) When designing cryptographic protocols and mechanisms, the following two steps are essential:

1. identify *all* assumptions in the protocol or mechanism design; and
2. for each assumption, determine the effect on the security objective if that assumption is violated.

1.11 Key establishment, management, and certification

This section gives a brief introduction to methodology for ensuring the secure distribution of keys for cryptographic purposes.

1.63 Definition *Key establishment* is any process whereby a shared secret key becomes available to two or more parties, for subsequent cryptographic use.

1.64 Definition *Key management* is the set of processes and mechanisms which support key establishment and the maintenance of ongoing keying relationships between parties, including replacing older keys with new keys as necessary.

Key establishment can be broadly subdivided into *key agreement* and *key transport*. Many and various protocols have been proposed to provide key establishment. Chapter 12 describes a number of these in detail. For the purpose of this chapter only a brief overview of issues related to key management will be given. Simple architectures based on symmetric-key and public-key cryptography along with the concept of certification will be addressed.

As noted in §1.5, a major issue when using symmetric-key techniques is the establishment of pairwise secret keys. This becomes more evident when considering a network of entities, any two of which may wish to communicate. Figure 1.15 illustrates a network consisting of 6 entities. The arrowed edges indicate the 15 possible two-party communications which could take place. Since each pair of entities wish to communicate, this small network requires the secure exchange of $\binom{6}{2} = 15$ key pairs. In a network with n entities, the number of secure key exchanges required is $\binom{n}{2} = \frac{n(n-1)}{2}$.

Handbook of Applied Cryptography by A. Menezes, P. van Oorschot and S. Vanstone.

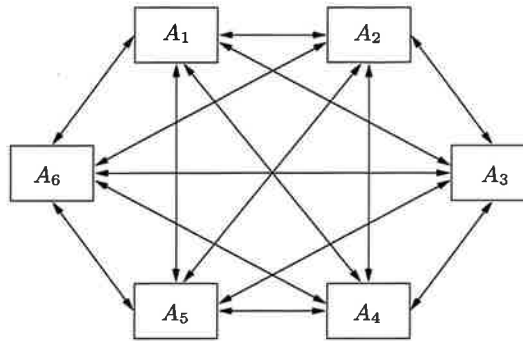


Figure 1.15: Keying relationships in a simple 6-party network.

The network diagram depicted in Figure 1.15 is simply the amalgamation of 15 two-party communications as depicted in Figure 1.7. In practice, networks are very large and the key management problem is a crucial issue. There are a number of ways to handle this problem. Two simplistic methods are discussed; one based on symmetric-key and the other on public-key techniques.

1.11.1 Key management through symmetric-key techniques

One solution which employs symmetric-key techniques involves an entity in the network which is trusted by all other entities. As in §1.8.3, this entity is referred to as a *trusted third party* (TTP). Each entity A_i shares a distinct symmetric key k_i with the TTP. These keys are assumed to have been distributed over a secured channel. If two entities subsequently wish to communicate, the TTP generates a key k (sometimes called a *session key*) and sends it encrypted under each of the fixed keys as depicted in Figure 1.16 for entities A_1 and A_5 .

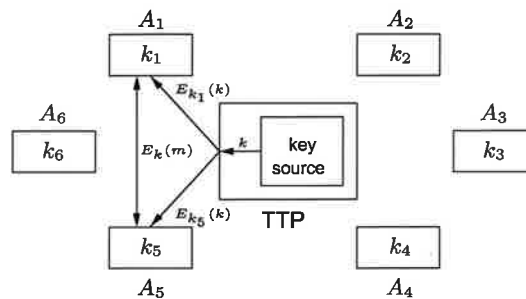


Figure 1.16: Key management using a trusted third party (TTP).

Advantages of this approach include:

1. It is easy to add and remove entities from the network.
2. Each entity needs to store only one long-term secret key.

Disadvantages include:

1. All communications require initial interaction with the TTP.
2. The TTP must store n long-term secret keys.

©1997 by CRC Press, Inc. — See accompanying notice at front of chapter.

3. The TTP has the ability to read all messages.
4. If the TTP is compromised, all communications are insecure.

1.11.2 Key management through public-key techniques

There are a number of ways to address the key management problem through public-key techniques. Chapter 13 describes many of these in detail. For the purpose of this chapter a very simple model is considered.

Each entity in the network has a public/private encryption key pair. The public key along with the identity of the entity is stored in a central repository called a *public file*. If an entity A_1 wishes to send encrypted messages to entity A_6 , A_1 retrieves the public key e_6 of A_6 from the public file, encrypts the message using this key, and sends the ciphertext to A_6 . Figure 1.17 depicts such a network.

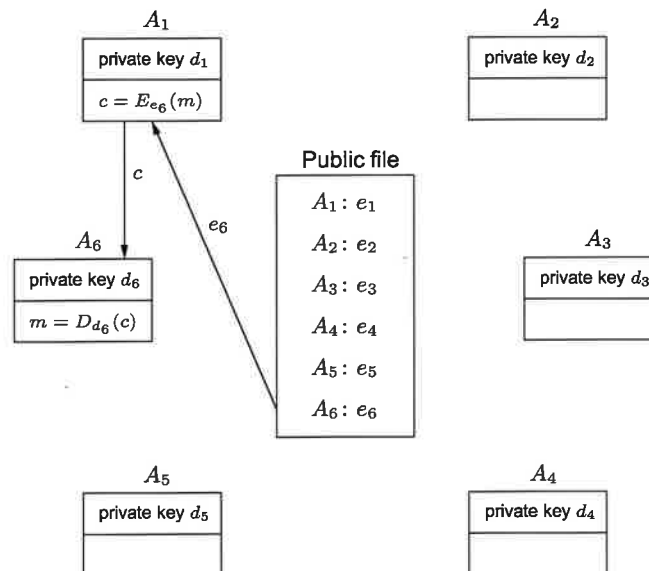


Figure 1.17: Key management using public-key techniques.

Advantages of this approach include:

1. No trusted third party is required.
2. The public file could reside with each entity.
3. Only n public keys need to be stored to allow secure communications between any pair of entities, assuming the only attack is that by a passive adversary.

The key management problem becomes more difficult when one must take into account an adversary who is *active* (i.e. an adversary who can alter the public file containing public keys). Figure 1.18 illustrates how an active adversary could compromise the key management scheme given above. (This is directly analogous to the attack in §1.8.2.) In the figure, the adversary alters the public file by replacing the public key e_6 of entity A_6 by the adversary's public key e^* . Any message encrypted for A_6 using the public key from the public file can be decrypted by only the adversary. Having decrypted and read the message, the

Handbook of Applied Cryptography by A. Menezes, P. van Oorschot and S. Vanstone.

adversary can now encrypt it using the public key of A_6 and forward the ciphertext to A_6 . A_1 however believes that only A_6 can decrypt the ciphertext c .

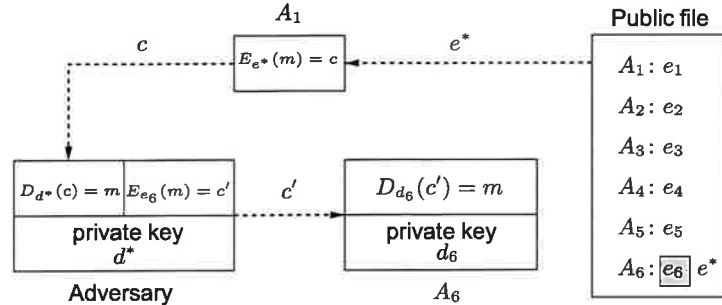


Figure 1.18: An impersonation of A_6 by an active adversary with public key e^* .

To prevent this type of attack, the entities may use a TTP to *certify* the public key of each entity. The TTP has a private signing algorithm S_T and a verification algorithm V_T (see §1.6) assumed to be known by all entities. The TTP carefully verifies the identity of each entity, and signs a message consisting of an identifier and the entity's authentic public key. This is a simple example of a *certificate*, binding the identity of an entity to its public key (see §1.11.3). Figure 1.19 illustrates the network under these conditions. A_1 uses the public key of A_6 only if the certificate signature verifies successfully.

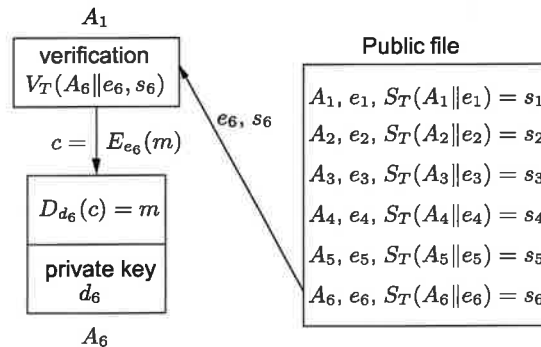


Figure 1.19: Authentication of public keys by a TTP. $\parallel$ denotes concatenation.

Advantages of using a TTP to maintain the integrity of the public file include:

1. It prevents an active adversary from impersonation on the network.
2. The TTP cannot monitor communications. Entities need trust the TTP only to bind identities to public keys properly.
3. Per-communication interaction with the public file can be eliminated if entities store certificates locally.

Even with a TTP, some concerns still remain:

1. If the signing key of the TTP is compromised, all communications become insecure.
2. All trust is placed with one entity.

1.11.3 Trusted third parties and public-key certificates

A trusted third party has been used in §1.8.3 and again here in §1.11. The trust placed on this entity varies with the way it is used, and hence motivates the following classification.

1.65 Definition A TTP is said to be *unconditionally trusted* if it is trusted on all matters. For example, it may have access to the secret and private keys of users, as well as be charged with the association of public keys to identifiers.

1.66 Definition A TTP is said to be *functionally trusted* if the entity is assumed to be honest and fair but it does not have access to the secret or private keys of users.

§1.11.1 provides a scenario which employs an unconditionally trusted TTP. §1.11.2 uses a functionally trusted TTP to maintain the integrity of the public file. A functionally trusted TTP could be used to register or certify users and contents of documents or, as in §1.8.3, as a judge.

Public-key certificates

The distribution of public keys is generally easier than that of symmetric keys, since secrecy is not required. However, the integrity (authenticity) of public keys is critical (recall §1.8.2).

A *public-key certificate* consists of a *data part* and a *signature part*. The data part consists of the name of an entity, the public key corresponding to that entity, possibly additional relevant information (e.g., the entity's street or network address, a validity period for the public key, and various other attributes). The signature part consists of the signature of a TTP over the data part.

In order for an entity B to verify the authenticity of the public key of an entity A , B must have an authentic copy of the public signature verification function of the TTP. For simplicity, assume that the authenticity of this verification function is provided to B by non-cryptographic means, for example by B obtaining it from the TTP in person. B can then carry out the following steps:

1. Acquire the public-key certificate of A over some unsecured channel, either from a central database of certificates, from A directly, or otherwise.
2. Use the TTP's verification function to verify the TTP's signature on A 's certificate.
3. If this signature verifies correctly, accept the public key in the certificate as A 's authentic public key; otherwise, assume the public key is invalid.

Before creating a public-key certificate for A , the TTP must take appropriate measures to verify the identity of A and the fact that the public key to be certificated actually belongs to A . One method is to require that A appear before the TTP with a conventional passport as proof of identity, and obtain A 's public key from A in person along with evidence that A knows the corresponding private key. Once the TTP creates a certificate for a party, the trust that all other entities have in the authenticity of the TTP's public key can be used transitively to gain trust in the authenticity of that party's public key, through acquisition and verification of the certificate.

1.12 Pseudorandom numbers and sequences

Random number generation is an important primitive in many cryptographic mechanisms. For example, keys for encryption transformations need to be generated in a manner which is

Handbook of Applied Cryptography by A. Menezes, P. van Oorschot and S. Vanstone.

unpredictable to an adversary. Generating a random key typically involves the selection of random numbers or bit sequences. Random number generation presents challenging issues. A brief introduction is given here with details left to Chapter 5.

Often in cryptographic applications, one of the following steps must be performed:

- (i) From a finite set of n elements (e.g., $\{1, 2, \dots, n\}$), select an element at random.
- (ii) From the set of all sequences (strings) of length m over some finite alphabet $\mathcal{A}$ of n symbols, select a sequence at random.
- (iii) Generate a random sequence (string) of symbols of length m over a set of n symbols.

It is not clear what exactly it means to *select at random* or *generate at random*. Calling a number random without a context makes little sense. Is the number 23 a random number? No, but if 49 identical balls labeled with a number from 1 to 49 are in a container, and this container mixes the balls uniformly, drops one ball out, and this ball happens to be labeled with the number 23, then one would say that 23 was generated randomly from a uniform distribution. The *probability* that 23 drops out is 1 in 49 or $\frac{1}{49}$.

If the number on the ball which was dropped from the container is recorded and the ball is placed back in the container and the process repeated 6 times, then a random sequence of length 6 defined on the alphabet $\mathcal{A} = \{1, 2, \dots, 49\}$ will have been generated. What is the chance that the sequence 17, 45, 1, 7, 23, 35 occurs? Since each element in the sequence has probability $\frac{1}{49}$ of occurring, the probability of the sequence 17, 45, 1, 7, 23, 35 occurring is

$$\frac{1}{49} \times \frac{1}{49} \times \frac{1}{49} \times \frac{1}{49} \times \frac{1}{49} \times \frac{1}{49} = \frac{1}{13841287201}.$$

There are precisely 13841287201 sequences of length 6 over the alphabet $\mathcal{A}$. If each of these sequences is written on one of 13841287201 balls and they are placed in the container (first removing the original 49 balls) then the chance that the sequence given above drops out is the same as if it were generated one ball at a time. Hence, (ii) and (iii) above are essentially the same statements.

Finding good methods to generate random sequences is difficult.

1.67 Example (*random sequence generator*) To generate a random sequence of 0's and 1's, a coin could be tossed with a head landing up recorded as a 1 and a tail as a 0. It is assumed that the coin is *unbiased*, which means that the probability of a 1 on a given toss is exactly $\frac{1}{2}$. This will depend on how well the coin is made and how the toss is performed. This method would be of little value in a system where random sequences must be generated quickly and often. It has no practical value other than to serve as an example of the idea of random number generation. $\square$

1.68 Example (*random sequence generator*) A *noise diode* may be used to produce random binary sequences. This is reasonable if one has some way to be convinced that the probability that a 1 will be produced on any given trial is $\frac{1}{2}$. Should this assumption be false, the sequence generated would not have been selected from a uniform distribution and so not all sequences of a given length would be equally likely. The only way to get some feeling for the reliability of this type of random source is to carry out statistical tests on its output. These are considered in Chapter 5. If the diode is a source of a uniform distribution on the set of all binary sequences of a given length, it provides an effective way to generate random sequences. $\square$

Since most *true sources* of random sequences (if there is such a thing) come from *physical means*, they tend to be either costly or slow in their generation. To overcome these

problems, methods have been devised to construct *pseudorandom sequences* in a deterministic manner from a shorter random sequence called a *seed*. The pseudorandom sequences appear to be generated by a truly random source to anyone not knowing the method of generation. Often the generation algorithm is known to all, but the seed is unknown except by the entity generating the sequence. A plethora of algorithms has been developed to generate pseudorandom bit sequences of various types. Many of these are completely unsuitable for cryptographic purposes and one must be cautious of claims by creators of such algorithms as to the random nature of the output.

1.13 Classes of attacks and security models

Over the years, many different types of attacks on cryptographic primitives and protocols have been identified. The discussion here limits consideration to attacks on encryption and protocols. Attacks on other cryptographic primitives will be given in appropriate chapters.

In §1.11 the roles of an active and a passive adversary were discussed. The attacks these adversaries can mount may be classified as follows:

1. A *passive attack* is one where the adversary only monitors the communication channel. A passive attacker only threatens confidentiality of data.
2. An *active attack* is one where the adversary attempts to delete, add, or in some other way alter the transmission on the channel. An active attacker threatens data integrity and authentication as well as confidentiality.

A passive attack can be further subdivided into more specialized attacks for deducing plaintext from ciphertext, as outlined in §1.13.1.

1.13.1 Attacks on encryption schemes

The objective of the following attacks is to systematically recover plaintext from ciphertext, or even more drastically, to deduce the decryption key.

1. A *ciphertext-only attack* is one where the adversary (or cryptanalyst) tries to deduce the decryption key or plaintext by only observing ciphertext. Any encryption scheme vulnerable to this type of attack is considered to be completely insecure.
2. A *known-plaintext attack* is one where the adversary has a quantity of plaintext and corresponding ciphertext. This type of attack is typically only marginally more difficult to mount.
3. A *chosen-plaintext attack* is one where the adversary chooses plaintext and is then given corresponding ciphertext. Subsequently, the adversary uses any information deduced in order to recover plaintext corresponding to previously unseen ciphertext.
4. An *adaptive chosen-plaintext attack* is a chosen-plaintext attack wherein the choice of plaintext may depend on the ciphertext received from previous requests.
5. A *chosen-ciphertext attack* is one where the adversary selects the ciphertext and is then given the corresponding plaintext. One way to mount such an attack is for the adversary to gain access to the equipment used for decryption (but not the decryption key, which may be securely embedded in the equipment). The objective is then to be able, without access to such equipment, to deduce the plaintext from (different) ciphertext.

6. An *adaptive chosen-ciphertext attack* is a chosen-ciphertext attack where the choice of ciphertext may depend on the plaintext received from previous requests.

Most of these attacks also apply to digital signature schemes and message authentication codes. In this case, the objective of the attacker is to forge messages or MACs, as discussed in Chapters 11 and 9, respectively.

1.13.2 Attacks on protocols

The following is a partial list of attacks which might be mounted on various protocols. Until a protocol is proven to provide the service intended, the list of possible attacks can never be said to be complete:

1. *known-key attack*. In this attack an adversary obtains some keys used previously and then uses this information to determine new keys.
2. *replay*. In this attack an adversary records a communication session and replays the entire session, or a portion thereof, at some later point in time.
3. *impersonation*. Here an adversary assumes the identity of one of the legitimate parties in a network.
4. *dictionary*. This is usually an attack against passwords. Typically, a password is stored in a computer file as the image of an unkeyed hash function. When a user logs on and enters a password, it is hashed and the image is compared to the stored value. An adversary can take a list of probable passwords, hash all entries in this list, and then compare this to the list of true encrypted passwords with the hope of finding matches.
5. *forward search*. This attack is similar in spirit to the dictionary attack and is used to decrypt messages. An example of this method was cited in Example 1.60.
6. *interleaving attack*. This type of attack usually involves some form of impersonation in an authentication protocol (see §12.9.1).

1.13.3 Models for evaluating security

The security of cryptographic primitives and protocols can be evaluated under several different models. The most practical security metrics are computational, provable, and ad hoc methodology, although the latter is often dangerous. The confidence level in the amount of security provided by a primitive or protocol based on computational or ad hoc security increases with time and investigation of the scheme. However, time is not enough if few people have given the method careful analysis.

(i) Unconditional security

The most stringent measure is an information-theoretic measure – whether or not a system has *unconditional security*. An adversary is assumed to have unlimited computational resources, and the question is whether or not there is enough information available to defeat the system. Unconditional security for encryption systems is called *perfect secrecy*. For perfect secrecy, the uncertainty in the plaintext, after observing the ciphertext, must be equal to the a priori uncertainty about the plaintext – observation of the ciphertext provides no information whatsoever to an adversary.

A necessary condition for a symmetric-key encryption scheme to be unconditionally secure is that the key be at least as long as the message. The one-time pad (§1.5.4) is an example of an unconditionally secure encryption algorithm. In general, encryption schemes

do not offer perfect secrecy, and each ciphertext character observed decreases the theoretical uncertainty in the plaintext and the encryption key. Public-key encryption schemes cannot be unconditionally secure since, given a ciphertext c , the plaintext can in principle be recovered by encrypting all possible plaintexts until c is obtained.

(ii) Complexity-theoretic security

An appropriate model of computation is defined and adversaries are modeled as having polynomial computational power. (They mount attacks involving time and space polynomial in the size of appropriate security parameters.) A proof of security relative to the model is then constructed. An objective is to design a cryptographic method based on the weakest assumptions possible anticipating a powerful adversary. Asymptotic analysis and usually also worst-case analysis is used and so care must be exercised to determine when proofs have practical significance. In contrast, polynomial attacks which are feasible under the model might, in practice, still be computationally infeasible.

Security analysis of this type, although not of practical value in all cases, may nonetheless pave the way to a better overall understanding of security. Complexity-theoretic analysis is invaluable for formulating fundamental principles and confirming intuition. This is like many other sciences, whose practical techniques are discovered early in the development, well before a theoretical basis and understanding is attained.

(iii) Provable security

A cryptographic method is said to be *provably secure* if the difficulty of defeating it can be shown to be essentially as difficult as solving a well-known and *supposedly* difficult (typically number-theoretic) problem, such as integer factorization or the computation of discrete logarithms. Thus, “provable” here means provable subject to assumptions.

This approach is considered by some to be as good a practical analysis technique as exists. Provable security may be considered part of a special sub-class of the larger class of computational security considered next.

(iv) Computational security

This measures the amount of computational effort required, by the best currently-known methods, to defeat a system; it must be assumed here that the system has been well-studied to determine which attacks are relevant. A proposed technique is said to be *computationally secure* if the perceived level of computation required to defeat it (using the best attack known) exceeds, by a comfortable margin, the computational resources of the hypothesized adversary.

Often methods in this class are related to hard problems but, unlike for provable security, no proof of equivalence is known. Most of the best known public-key and symmetric-key schemes in current use are in this class. This class is sometimes also called *practical security*.

(v) Ad hoc security

This approach consists of any variety of convincing arguments that every successful attack requires a resource level (e.g., time and space) greater than the fixed resources of a perceived adversary. Cryptographic primitives and protocols which survive such analysis are said to have *heuristic security*, with security here typically in the computational sense.

Primitives and protocols are usually designed to counter standard attacks such as those given in §1.13. While perhaps the most commonly used approach (especially for protocols), it is, in some ways, the least satisfying. Claims of security generally remain questionable and unforeseen attacks remain a threat.

Handbook of Applied Cryptography by A. Menezes, P. van Oorschot and S. Vanstone.

1.13.4 Perspective for computational security

To evaluate the security of cryptographic schemes, certain quantities are often considered.

1.69 Definition The *work factor* W_d is the minimum amount of work (measured in appropriate units such as elementary operations or clock cycles) required to compute the private key d given the public key e , or, in the case of symmetric-key schemes, to determine the secret key k . More specifically, one may consider the work required under a ciphertext-only attack given n ciphertexts, denoted $W_d(n)$.

If W_d is t years, then for sufficiently large t the cryptographic scheme is, for all practical purposes, a secure system. To date no public-key system has been found where one can prove a sufficiently large lower bound on the work factor W_d . The best that is possible to date is to rely on the following as a basis for security.

1.70 Definition The *historical work factor* $\overline{W}_d$ is the minimum amount of work required to compute the private key d from the public key e using the best known algorithms at a given point in time.

The historical work factor $\overline{W}_d$ varies with time as algorithms and technology improve. It corresponds to computational security, whereas W_d corresponds to the true security level, although this typically cannot be determined.

How large is large?

§1.4 described how the designer of an encryption system tries to create a scheme for which the best approach to breaking it is through exhaustive search of the key space. The key space must then be large enough to make an exhaustive search completely infeasible. An important question then is “How large is large?”. In order to gain some perspective on the magnitude of numbers, Table 1.2 lists various items along with an associated magnitude.

Reference	Magnitude
Seconds in a year	$\approx 3 \times 10^7$
Age of our solar system (years)	$\approx 6 \times 10^9$
Seconds since creation of solar system	$\approx 2 \times 10^{17}$
Clock cycles per year, 50 MHz computer	$\approx 1.6 \times 10^{15}$
Binary strings of length 64	$2^{64} \approx 1.8 \times 10^{19}$
Binary strings of length 128	$2^{128} \approx 3.4 \times 10^{38}$
Binary strings of length 256	$2^{256} \approx 1.2 \times 10^{77}$
Number of 75-digit prime numbers	$\approx 5.2 \times 10^{72}$
Electrons in the universe	$\approx 8.37 \times 10^{77}$

Table 1.2: Reference numbers comparing relative magnitudes.

Some powers of 10 are referred to by prefixes. For example, high-speed modern computers are now being rated in terms of *teraflops* where a teraflop is 10^{12} floating point operations per second. Table 1.3 provides a list of commonly used prefixes.

Prefix	Symbol	Magnitude	Prefix	Symbol	Magnitude
exa	E	10^{18}	deci	d	10^{-1}
peta	P	10^{15}	centi	c	10^{-2}
tera	T	10^{12}	milli	m	10^{-3}
giga	G	10^9	micro	μ	10^{-6}
mega	M	10^6	nano	n	10^{-9}
kilo	k	10^3	pico	p	10^{-12}
hecto	h	10^2	femto	f	10^{-15}
deca	da	10	atto	a	10^{-18}

Table 1.3: Prefixes used for various powers of 10.

1.14 Notes and further references

§1.1

Kahn [648] gives a thorough, comprehensive, and non-technical history of cryptography, published in 1967. Feistel [387] provides an early exposition of block cipher ideas. The original specification of DES is the 1977 U.S. Federal Information Processing Standards Publication 46 [396]. Public-key cryptography was introduced by Diffie and Hellman [345]. The first concrete realization of a public-key encryption scheme was the knapsack scheme by Merkle and Hellman [857]. The RSA public-key encryption and signature scheme is due to Rivest, Shamir, and Adleman [1060], while the ElGamal public-key encryption and signature schemes are due to ElGamal [368]. The two digital signature standards, ISO/IEC 9796 [596] and the Digital Signature Standard [406], are discussed extensively in Chapter 11.

Cryptography has used specialized areas of mathematics such as number theory to realize very practical mechanisms such as public-key encryption and digital signatures. Such usage was not conceived as possible a mere twenty years ago. The famous mathematician, Hardy [539], went as far as to boast about its lack of utility:

“... both Gauss and lesser mathematicians may be justified in rejoicing that there is one science at any rate, and that their own, whose very remoteness from ordinary human activities should keep it gentle and clean.”

§1.2

This section was inspired by the foreword to the book *Contemporary Cryptology, The Science of Information Integrity*, edited by Simmons [1143]. The handwritten signature came into the British legal system in the seventeenth century as a means to provide various functions associated with information security. See Chapter 9 of Meyer and Matyas [859] for details.

This book only considers cryptography as it applies to information in digital form. Chapter 9 of Beker and Piper [84] provides an introduction to the encryption of analogue signals, in particular, speech. Although in many cases physical means are employed to facilitate privacy, cryptography plays the major role. Physical means of providing privacy include fiber optic communication links, spread spectrum technology, TEMPEST techniques, and

Handbook of Applied Cryptography by A. Menezes, P. van Oorschot and S. Vanstone.

tamper-resistant hardware. *Steganography* is that branch of information privacy which attempts to obscure the existence of data through such devices as invisible inks, secret compartments, the use of subliminal channels, and the like. Kahn [648] provides an historical account of various steganographic techniques.

Excellent introductions to cryptography can be found in the articles by Diffie and Hellman [347], Massey [786], and Rivest [1054]. A concise and elegant way to describe cryptography was given by Rivest [1054]: *Cryptography is about communication in the presence of adversaries*. The taxonomy of cryptographic primitives (Figure 1.1) was derived from the classification given by Bosselaers, Govaerts, and Vandewalle [175].

§1.3

The theory of functions is fundamental in modern mathematics. The term *range* is often used in place of image of a function. The latter, being more descriptive, is preferred. An alternate term for one-to-one is *injective*; an alternate term for onto is *surjective*.

One-way functions were introduced by Diffie and Hellman [345]. A more extensive history is given on page 377. Trapdoor one-way functions were first postulated by Diffie and Hellman [345] and independently by Merkle [850] as a means to obtain public-key encryption schemes; several candidates are given in Chapter 8.

§1.4

The basic concepts of cryptography are treated quite differently by various authors, some being more technical than others. Brassard [192] provides a concise, lucid, and technically accurate account. Schneier [1094] gives a less technical but very accessible introduction. Salomaa [1089], Stinson [1178], and Rivest [1054] present more mathematical approaches. Davies and Price [308] provide a very readable presentation suitable for the practitioner.

The comparison of an encryption scheme to a resettable combination lock is from Diffie and Hellman [347]. Kerckhoffs' desiderata [668] were originally stated in French. The translation stated here is given in Kahn [648]. Shannon [1121] also gives desiderata for encryption schemes.

§1.5

Symmetric-key encryption has a very long history, as recorded by Kahn [648]. Most systems invented prior to the 1970s are now of historical interest only. Chapter 2 of Denning [326] is also a good source for many of the more well known schemes such as the Caesar cipher, Vigenère and Beaufort ciphers, rotor machines (Enigma and Hagelin), running key ciphers, and so on; see also Davies and Price [308] and Konheim [705]. Beker and Piper [84] give an indepth treatment, including cryptanalysis of several of the classical systems used in World War II. Shannon's paper [1121] is considered the seminal work on secure communications. It is also an excellent source for descriptions of various well-known historical symmetric-key ciphers.

Simple substitution and transposition ciphers are the focus of §1.5. Hill ciphers [557], a class of substitution ciphers which substitute blocks using matrix methods, are covered in Example 7.52. The idea of confusion and diffusion (Remark 1.36) was introduced by Shannon [1121].

Kahn [648] gives 1917 as the date when Vernam discovered the cipher which bears Vernam's name, however, Vernam did not publish the result until 1926 [1222]; see page 274 for further discussion. Massey [786] states that reliable sources have suggested that the Moscow-Washington hot-line (channel for very high level communications) is no longer secured with a one-time pad, which has been replaced by a symmetric-key cipher requiring a much shorter key. This change would indicate that confidence and understanding in the

ability to construct very strong symmetric-key encryption schemes exists. The one-time pad seems to have been used extensively by Russian agents operating in foreign countries. The highest ranking Russian agent ever captured in the United States was Rudolph Abel. When apprehended in 1957 he had in his possession a booklet the size of a postage stamp ($1\frac{7}{8} \times \frac{7}{8} \times \frac{7}{8}$ inches) containing a one-time key; see Kahn [648, p.664].

§1.6

The concept of a digital signature was introduced by Diffie and Hellman [345] and independently by Merkle [850]. The first practical realization of a digital signature scheme appeared in the paper by Rivest, Shamir, and Adleman [1060]. Rabin [1022] (see also [1023]) also claims to have independently discovered RSA but did not publish the result.

Most introductory sources for digital signatures stress digital signatures with message recovery coming from a public-key encryption system. Mitchell, Piper, and Wild [882] give a good general treatment of the subject. Stinson [1178] provides a similar elementary but general introduction. Chapter 11 generalizes the definition of a digital signature by allowing randomization. The scheme described in §1.8 is referred to as *deterministic*. Many other types of digital signatures with specific properties have been created, such as blind signatures, undeniable signatures, and failstop signatures (see Chapter 11).

§1.7

Much effort has been devoted to developing a theory of authentication. At the forefront of this is Simmons [1144], whose contributions are nicely summarized by Massey [786]. For a more concrete example of the necessity for authentication without secrecy, see the article by Simmons [1146].

§1.8

1976 marked a major turning point in the history of cryptography. In several papers that year, Diffie and Hellman introduced the idea of public-key cryptography and gave concrete examples of how such a scheme might be realized. The first paper on public-key cryptography was “Multiuser cryptographic techniques” by Diffie and Hellman [344], presented at the National Computer Conference in June of 1976. Although the authors were not satisfied with the examples they cited, the concept was made clear. In their landmark paper, Diffie and Hellman [345] provided a more comprehensive account of public-key cryptography and described the first viable method to realize this elegant concept. Another good source for the early history and development of the subject is Diffie [343]. Nechvatal [922] also provides a broad survey of public-key cryptography.

Merkle [849, 850] independently discovered public-key cryptography, illustrating how this concept could be realized by giving an elegant and ingenious example now commonly referred to as the *Merkle puzzle scheme*. Simmons [1144, p.412] notes the first reported application of public-key cryptography was fielded by Sandia National Laboratories (U.S.) in 1978.

§1.9

Much of the early work on cryptographic hash functions was done by Merkle [850]. The most comprehensive current treatment of the subject is by Preneel [1004].

§1.10

A large number of successful cryptanalytic attacks on systems claiming security are due to protocol failure. An overview of this area is given by Moore [899], including classifications of protocol failures and design principles.

Handbook of Applied Cryptography by A. Menezes, P. van Oorschot and S. Vanstone.

§1.11

One approach to distributing public-keys is the so-called *Merkle channel* (see Simmons [1144, p.387]). Merkle proposed that public keys be distributed over so many independent public channels (newspaper, radio, television, etc.) that it would be improbable for an adversary to compromise all of them.

In 1979 Kohnfelder [702] suggested the idea of using *public-key certificates* to facilitate the distribution of public keys over unsecured channels, such that their authenticity can be verified. Essentially the same idea, but by on-line requests, was proposed by Needham and Schroeder (see Wilkes [1244]).

A provably secure key agreement protocol has been proposed whose security is based on the Heisenberg uncertainty principle of quantum physics. The security of so-called *quantum cryptography* does not rely upon any complexity-theoretic assumptions. For further details on quantum cryptography, consult Chapter 6 of Brassard [192], and Bennett, Brassard, and Ekert [115].

§1.12

For an introduction and detailed treatment of many pseudorandom sequence generators, see Knuth [692]. Knuth cites an example of a complex scheme to generate random numbers which on closer analysis is shown to produce numbers which are far from random, and concludes: *...random numbers should not be generated with a method chosen at random.*

§1.13

The seminal work of Shannon [1121] on secure communications, published in 1949, remains as one of the best introductions to both practice and theory, clearly presenting many of the fundamental ideas including redundancy, entropy, and unicity distance. Various models under which security may be examined are considered by Rueppel [1081], Simmons [1144], and Preneel [1003], among others; see also Goldwasser [476].

This is a Chapter from the **Handbook of Applied Cryptography**, by A. Menezes, P. van Oorschot, and S. Vanstone, CRC Press, 1996.

For further information, see www.cacr.math.uwaterloo.ca/hac

CRC Press has granted the following specific permissions for the electronic version of this book:

Permission is granted to retrieve, print and store a single copy of this chapter for personal use. This permission does not extend to binding multiple chapters of the book, photocopying or producing copies for other than personal use of the person creating the copy, or making electronic copies available for retrieval by others without prior permission in writing from CRC Press.

Except where over-ridden by the specific permission above, the standard copyright notice from CRC Press applies to this electronic version:

Neither this book nor any part may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, microfilming, and recording, or by any information storage or retrieval system, without prior permission in writing from the publisher.

The consent of CRC Press does not extend to copying for general distribution, for promotion, for creating new works, or for resale. Specific permission must be obtained in writing from CRC Press for such copying.

©1997 by CRC Press, Inc.

Chapter 10

Identification and Entity Authentication

Contents in Brief

10.1 Introduction	385
10.2 Passwords (weak authentication)	388
10.3 Challenge-response identification (strong authentication)	397
10.4 Customized and zero-knowledge identification protocols	405
10.5 Attacks on identification protocols	417
10.6 Notes and further references	420

10.1 Introduction

This chapter considers techniques designed to allow one party (the *verifier*) to gain assurances that the identity of another (the *claimant*) is as declared, thereby preventing impersonation. The most common technique is by the verifier checking the correctness of a message (possibly in response to an earlier message) which demonstrates that the claimant is in possession of a secret associated by design with the genuine party. Names for such techniques include *identification*, *entity authentication*, and (less frequently) *identity verification*. Related topics addressed elsewhere include message authentication (data origin authentication) by symmetric techniques (Chapter 9) and digital signatures (Chapter 11), and authenticated key establishment (Chapter 12).

A major difference between entity authentication and message authentication (as provided by digital signatures or MACs) is that message authentication itself provides no timeliness guarantees with respect to when a message was created, whereas entity authentication involves corroboration of a claimant's identity through actual communications with an associated verifier during execution of the protocol itself (i.e., in *real-time*, while the verifying entity awaits). Conversely, entity authentication typically involves no meaningful message other than the claim of being a particular entity, whereas message authentication does. Techniques which provide both entity authentication and key establishment are deferred to Chapter 12; in some cases, key establishment is essentially message authentication where the message is the key.

Chapter outline

The remainder of §10.1 provides introductory material. §10.2 discusses identification schemes involving fixed passwords including Personal Identification Numbers (PINs), and providing so-called weak authentication; one-time password schemes are also considered. §10.3 considers techniques providing so-called strong authentication, including challenge-response protocols based on both symmetric and public-key techniques. It includes discussion of time-variant parameters (TVPs), which may be used in entity authentication protocols and to provide uniqueness or timeliness guarantees in message authentication. §10.4 examines customized identification protocols based on or motivated by zero-knowledge techniques. §10.5 considers attacks on identification protocols. §10.6 provides references and further chapter notes.

10.1.1 Identification objectives and applications

The general setting for an identification protocol involves a *prover* or *claimant* A and a *verifier* B . The verifier is presented with, or presumes beforehand, the purported identity of the claimant. The goal is to corroborate that the identity of the claimant is indeed A , i.e., to provide entity authentication.

10.1 Definition *Entity authentication* is the process whereby one party is assured (through acquisition of corroborative evidence) of the identity of a second party involved in a protocol, and that the second has actually participated (i.e., is active at, or immediately prior to, the time the evidence is acquired).

10.2 Remark (*identification terminology*) The terms *identification* and *entity authentication* are used synonymously throughout this book. Distinction is made between weak, strong, and zero-knowledge based authentication. Elsewhere in the literature, sometimes identification implies only a claimed or stated identity whereas entity authentication suggests a corroborated identity.

(i) Objectives of identification protocols

From the point of view of the verifier, the outcome of an entity authentication protocol is either *acceptance* of the claimant's identity as authentic (completion with acceptance), or *termination without acceptance* (rejection). More specifically, the objectives of an identification protocol include the following.

1. In the case of honest parties A and B , A is able to successfully authenticate itself to B , i.e., B will complete the protocol having accepted A 's identity.
2. (*transferability*) B cannot reuse an identification exchange with A so as to successfully impersonate A to a third party C .
3. (*impersonation*) The probability is negligible that any party C distinct from A , carrying out the protocol and playing the role of A , can cause B to complete and accept A 's identity. Here *negligible* typically means "is so small that it is not of practical significance"; the precise definition depends on the application.
4. The previous points remain true even if: a (polynomially) large number of previous authentications between A and B have been observed; the adversary C has participated in previous protocol executions with either or both A and B ; and multiple instances of the protocol, possibly initiated by C , may be run simultaneously.

The idea of zero-knowledge-based protocols is that protocol executions do not even reveal any partial information which makes C 's task any easier whatsoever.

An identification (or entity authentication) protocol is a “real-time” process in the sense that it provides an assurance that the party being authenticated is operational at the time of protocol execution – that party is taking part, having carried out some action since the start of the protocol execution. Identification protocols provide assurances only at the particular instant in time of successful protocol completion. If ongoing assurances are required, additional measures may be necessary; see §10.5.

(ii) Basis of identification

Entity authentication techniques may be divided into three main categories, depending on which of the following the security is based:

1. *something known*. Examples include standard passwords (sometimes used to derive a symmetric key), Personal Identification Numbers (PINs), and the secret or private keys whose knowledge is demonstrated in challenge-response protocols.
2. *something possessed*. This is typically a physical accessory, resembling a passport in function. Examples include magnetic-striped cards, *chipcards* (plastic cards the size of credit cards, containing an embedded microprocessor or integrated circuit; also called *smart cards* or *IC cards*), and hand-held customized calculators (*password generators*) which provide time-variant passwords.
3. *something inherent* (to a human individual). This category includes methods which make use of human physical characteristics and involuntary actions (*biometrics*), such as handwritten signatures, fingerprints, voice, retinal patterns, hand geometries, and dynamic keyboarding characteristics. These techniques are typically non-cryptographic and are not discussed further here.

(iii) Applications of identification protocols

One of the primary purposes of identification is to facilitate access control to a resource, when an access privilege is linked to a particular identity (e.g., local or remote access to computer accounts; withdrawals from automated cash dispensers; communications permissions through a communications port; access to software applications; physical entry to restricted areas or border crossings). A password scheme used to allow access to a user's computer account may be viewed as the simplest instance of an *access control matrix*: each resource has a list of identities associated with it (e.g., a computer account which authorized entities may access), and successful corroboration of an identity allows access to the authorized resources as listed for that entity. In many applications (e.g., cellular telephony) the motivation for identification is to allow resource usage to be tracked to identified entities, to facilitate appropriate billing. Identification is also typically an inherent requirement in authenticated key establishment protocols (see Chapter 12).

10.1.2 Properties of identification protocols

Identification protocols may have many properties. Properties of interest to users include:

1. *reciprocity of identification*. Either one or both parties may corroborate their identities to the other, providing, respectively, *unilateral* or *mutual* identification. Some techniques, such as fixed-password schemes, may be susceptible to an entity posing as a verifier simply in order to capture a claimant's password.
2. *computational efficiency*. The number of operations required to execute a protocol.

Handbook of Applied Cryptography by A. Menezes, P. van Oorschot and S. Vanstone.

3. *communication efficiency*. This includes the number of passes (message exchanges) and the bandwidth required (total number of bits transmitted).
More subtle properties include:
4. *real-time involvement of a third party (if any)*. Examples of third parties include an on-line *trusted* third party to distribute common symmetric keys to communicating entities for authentication purposes; and an on-line (untrusted) directory service for distributing public-key certificates, supported by an off-line certification authority (see Chapter 13).
5. *nature of trust required in a third party (if any)*. Examples include trusting a third party to correctly authenticate and bind an entity's name to a public key; and trusting a third party with knowledge of an entity's private key.
6. *nature of security guarantees*. Examples include provable security and zero-knowledge properties (see §10.4.1).
7. *storage of secrets*. This includes the location and method used (e.g., software only, local disks, hardware tokens, etc.) to store critical keying material.

Relation between identification and signature schemes

Identification schemes are closely related to, but simpler than, digital signature schemes, which involve a variable message and typically provide a non-repudiation feature allowing disputes to be resolved by judges after the fact. For identification schemes, the semantics of the message are essentially fixed – a claimed identity at the current instant in time. The claim is either corroborated or rejected immediately, with associated privileges or access either granted or denied in real time. Identifications do not have “lifetimes” as signatures do¹ – disputes need not typically be resolved afterwards regarding a prior identification, and attacks which may become feasible in the future do not affect the validity of a prior identification. In some cases, identification schemes may also be converted to signature schemes using a standard technique (see Note 10.30).

10.2 Passwords (weak authentication)

Conventional password schemes involve time-invariant passwords, which provide so-called *weak authentication*. The basic idea is as follows. A *password*, associated with each user (entity), is typically a string of 6 to 10 or more characters the user is capable of committing to memory. This serves as a shared secret between the user and system. (Conventional password schemes thus fall under the category of symmetric-key techniques providing unilateral authentication.) To gain access to a system resource (e.g., computer account, printer, or software application), the user enters a (userid, password) pair, and explicitly or implicitly specifies a resource; here *userid* is a claim of identity, and *password* is the evidence supporting the claim. The system checks that the password matches corresponding data it holds for that userid, and that the stated identity is authorized to access the resource. Demonstration of knowledge of this secret (by revealing the password itself) is accepted by the system as corroboration of the entity's identity.

Various password schemes are distinguished by the means by which information allowing password verification is stored within the system, and the method of verification. The collection of ideas presented in the following sections motivate the design decisions

¹Some identification techniques involve, as a by-product, the granting of *tickets* which provide time-limited access to specified resources (see Chapter 13).

made in typical password schemes. A subsequent section summarizes the standard attacks these designs counteract. Threats which must be guarded against include: password disclosure (outside of the system) and line eavesdropping (within the system), both of which allow subsequent replay; and password guessing, including dictionary attacks.

10.2.1 Fixed password schemes: techniques

(i) Stored password files

The most obvious approach is for the system to store user passwords cleartext in a system password file, which is both read- and write-protected (e.g., via operating system access control privileges). Upon password entry by a user, the system compares the entered password to the password file entry for the corresponding userid; employing no secret keys or cryptographic primitives such as encryption, this is classified as a non-cryptographic technique. A drawback of this method is that it provides no protection against privileged insiders or *superusers* (special userids which have full access privileges to system files and resources). Storage of the password file on backup media is also a security concern, since the file contains cleartext passwords.

(ii) “Encrypted” password files

Rather than storing a cleartext user password in a (read- and write-protected) password file, a one-way function of each user password is stored in place of the password itself (see Figure 10.1). To verify a user-entered password, the system computes the one-way function of the entered password, and compares this to the stored entry for the stated userid. To preclude attacks suggested in the preceding paragraph, the password file need now only be write-protected.

10.3 Remark (*one-way function vs. encryption*) For the purpose of protecting password files, the use of a one-way function is generally preferable to reversible encryption; reasons include those related to export restrictions, and the need for keying material. However, in both cases, for historical reasons, the resulting values are typically referred to as “encrypted” passwords. Protecting passwords by either method before transmission over public communications lines addresses the threat of compromise of the password itself, but alone does not preclude disclosure or replay of the transmission (cf. Protocol 10.6).

(iii) Password rules

Since dictionary attacks (see §10.2.2(iii)) are successful against predictable passwords, some systems impose “password rules” to discourage or prevent users from using “weak” passwords. Typical password rules include a lower bound on the password length (e.g., 8 or 12 characters); a requirement for each password to contain at least one character from each of a set of categories (e.g., uppercase, numeric, non-alphanumeric); or checks that candidate passwords are not found in on-line or available dictionaries, and are not composed of account-related information such as userids or substrings thereof.

Knowing which rules are in effect, an adversary may use a modified dictionary attack strategy taking into account the rules, and targeting the weakest form of passwords which nonetheless satisfy the rules. The objective of password rules is to increase the entropy (rather than just the length) of user passwords beyond the reach of dictionary and exhaustive search attacks. *Entropy* here refers to the uncertainty in a password (cf. §2.2.1); if all passwords are equally probable, then the entropy is maximal and equals the base-2 logarithm of the number of possible passwords.

Handbook of Applied Cryptography by A. Menezes, P. van Oorschot and S. Vanstone.

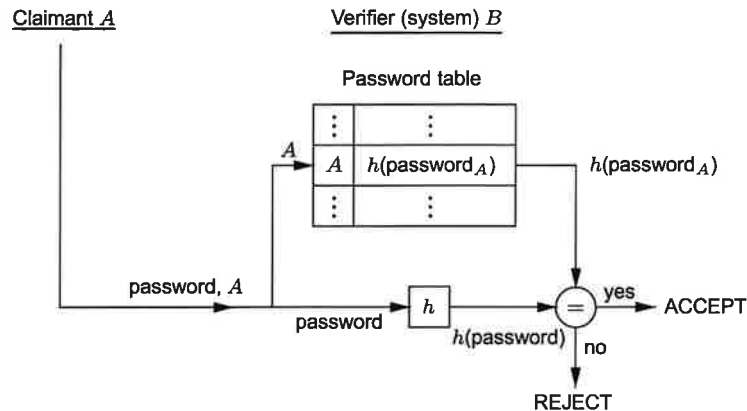


Figure 10.1: Use of one-way function for password-checking.

Another procedural technique intended to improve password security is *password aging*. A time period is defined limiting the lifetime of each particular password (e.g., 30 or 90 days). This requires that passwords be changed periodically.

(iv) Slowing down the password mapping

To slow down attacks which involve testing a large number of trial passwords (see §10.2.2), the password verification function (e.g., one-way function) may be made more computationally intensive, for example, by iterating a simpler function $t > 1$ times, with the output of iteration i used as the input for iteration $i + 1$. The total number of iterations must be restricted so as not to impose a noticeable or unreasonable delay for legitimate users. Also, the iterated function should be such that the iterated mapping does not result in a final range space whose entropy is significantly decimated.

(v) Salting passwords

To make dictionary attacks less effective, each password, upon initial entry, may be augmented with a t -bit random string called a *salt* (it alters the “flavor” of the password; cf. §10.2.3) before applying the one-way function. Both the hashed password and the salt are recorded in the password file. When the user subsequently enters a password, the system looks up the salt, and applies the one-way function to the entered password, as altered or augmented by the salt. The difficulty of exhaustive search on any particular user’s password is unchanged by salting (since the salt is given in cleartext in the password file); however, salting increases the complexity of a dictionary attack against a large set of passwords simultaneously, by requiring the dictionary to contain 2^t variations of each trial password, implying a larger memory requirement for storing an encrypted dictionary, and correspondingly more time for its preparation. Note that with salting, two users who choose the same password have different entries in the system password file. In some systems, it may be appropriate to use an entity’s userid itself as salt.

(vi) Passphrases

To allow greater entropy without stepping beyond the memory capacity of human users, passwords may be extended to *passphrases*; in this case, the user types in a phrase or sentence rather than a short “word”. The passphrase is hashed down to a fixed-size value, which plays the same role as a password; here, it is important that the passphrase is not simply trun-

cated by the system, as passwords are in some systems. The idea is that users can remember phrases easier than random character sequences. If passwords resemble English text, then since each character contains only about 1.5 bits of entropy (Fact 7.67), a passphrase provides greater security through increased entropy than a short password. One drawback is the additional typing requirement.

10.2.2 Fixed password schemes: attacks

(i) Replay of fixed passwords

A weakness of schemes using fixed, reusable passwords (i.e., the basic scheme of §10.2), is the possibility that an adversary learns a user's password by observing it as it is typed in (or from where it may be written down). A second security concern is that user-entered passwords (or one-way hashes thereof) are transmitted in cleartext over the communications line between the user and the system, and are also available in cleartext temporarily during system verification. An eavesdropping adversary may record this data, allowing subsequent impersonation.

Fixed password schemes are thus of use when the password is transmitted over trusted communications lines safe from monitoring, but are not suitable in the case that passwords are transmitted over open communications networks. For example, in Figure 10.1, the claimant *A* may be a user logging in from home over a telephone modem, to a remote office site *B* two (or two thousand) miles away; the cleartext password might then travel over an unsecured telephone network (including possibly a wireless link), subject to eavesdropping.

In the case that remote identity verification is used for access to a local resource, e.g., an automated cash dispenser with on-line identity verification, the system response (accept/reject) must be protected in addition to the submitted password, and must include variability to prevent trivial replay of a time-invariant accept response.

(ii) Exhaustive password search

A very naive attack involves an adversary simply (randomly or systematically) trying passwords, one at a time, on the actual verifier, in hope that the correct password is found. This may be countered by ensuring passwords are chosen from a sufficiently large space, limiting the number of invalid (on-line) attempts allowed within fixed time periods, and slowing down the password mapping or login-process itself as in §10.2.1(iv). *Off-line attacks*, involving a (typically large) computation which does not require interacting with the actual verifier until a final stage, are of greater concern; these are now considered.

Given a password file containing one-way hashes of user passwords, an adversary may attempt to defeat the system by testing passwords one at a time, and comparing the one-way hash of each to passwords in the encrypted password file (see §10.2.1(ii)). This is theoretically possible since both the one-way mapping and the (guessed) plaintext are known. (This could be precluded by keeping any or all of the details of the one-way mapping or the password file itself secret, but it is not considered prudent to base the security of the system on the assumption that such details remain secret forever.) The feasibility of the attack depends on the number of passwords that need be checked before a match is expected (which itself depends on the number of possible passwords), and the time required to test each (see Example 10.4, Table 10.1, and Table 10.2). The latter depends on the password mapping used, its implementation, the instruction execution time of the host processor, and the number of processors available (note exhaustive search is parallelizable). The time required to actually compare the image of each trial password to all passwords in a password file is typically negligible.

10.4 Example (password entropy) Suppose passwords consist of strings of 7-bit ASCII characters. Each has a numeric value in the range 0-127. (When 8-bit characters are used, values 128-255 compose the *extended character set*, generally inaccessible from standard keyboards.) ASCII codes 0-31 are reserved for control characters; 32 is a space character; 33-126 are keyboard-accessible printable characters; and 127 is a special character. Table 10.1 gives the number of distinct n -character passwords composed of typical combinations of characters, indicating an upper bound on the security of such password spaces. $\square$

$\rightarrow c$ $\downarrow n$	26 (lowercase)	36 (lowercase alphanumeric)	62 (mixed case alphanumeric)	95 (keyboard characters)
5	23.5	25.9	29.8	32.9
6	28.2	31.0	35.7	39.4
7	32.9	36.2	41.7	46.0
8	37.6	41.4	47.6	52.6
9	42.3	46.5	53.6	59.1
10	47.0	51.7	59.5	65.7

Table 10.1: Bitsize of password space for various character combinations. The number of n -character passwords, given c choices per character, is c^n . The table gives the base-2 logarithm of this number of possible passwords.

$\rightarrow c$ $\downarrow n$	26 (lowercase)	36 (lowercase alphanumeric)	62 (mixed case alphanumeric)	95 (keyboard characters)
5	0.67 hr	3.4 hr	51 hr	430 hr
6	17 hr	120 hr	130 dy	4.7 yr
7	19 dy	180 dy	22 yr	440 yr
8	1.3 yr	18 yr	1400 yr	42000 yr
9	34 yr	640 yr	86000 yr	4.0×10^6 yr
10	890 yr	23000 yr	5.3×10^6 yr	3.8×10^8 yr

Table 10.2: Time required to search entire password space. The table gives the time T (in hours, days, or years) required to search or pre-compute over the entire specified spaces using a single processor (cf. Table 10.1). $T = c^n \cdot t \cdot y$, where t is the number of times the password mapping is iterated, and y the time per iteration, for $t = 25$, $y = 1/(125\ 000)$ sec. (This approximates the UNIX crypt command on a high-end PC performing DES at 1.0 Mbytes/s – see §10.2.3.)

(iii) Password-guessing and dictionary attacks

To improve upon the expected probability of success of an exhaustive search, rather than searching through the space of all possible passwords, an adversary may search the space in order of decreasing (expected) probability. While ideally arbitrary strings of n characters would be equiprobable as user-selected passwords, most (unrestricted) users select passwords from a small subset of the full password space (e.g., short passwords; dictionary words; proper names; lowercase strings). Such weak passwords with low entropy are easily guessed; indeed, studies indicate that a large fraction of user-selected passwords are found in typical (intermediate) dictionaries of only 150 000 words, while even a large dictionary of 250 000 words represents only a tiny fraction of all possible n -character passwords (see Table 10.1).

Passwords found in any on-line or available list of words may be uncovered by an adversary who tries all words in this list, using a so-called *dictionary attack*. Aside from traditional dictionaries as noted above, on-line dictionaries of words from foreign languages, or

on specialized topics such as music, film, etc. are available. For efficiency in repeated use by an adversary, an “encrypted” (hashed) list of dictionary or high-probability passwords may be created and stored on disk or tape; password images from system password files may then be collected, ordered (using a sorting algorithm or conventional hashing), and then compared to entries in the encrypted dictionary. Dictionary-style attacks are not generally successful at finding a particular user’s password, but find many passwords in most systems.

10.2.3 Case study – UNIX passwords

The UNIX² operating system provides a widely known, historically important example of a fixed password system, implementing many of the ideas of §10.2.1. A UNIX password file contains a one-way function of user passwords computed as follows: each user password serves as the key to encrypt a known plaintext (64 zero-bits). This yields a one-way function of the key, since only the user (aside from the system, temporarily during password verification) knows the password. For the encryption algorithm, a minor modification of DES (§7.4) is used, as described below; variations may appear in products outside of the USA. The technique described relies on the conjectured property that DES is resistant to known-plaintext attacks – given cleartext and the corresponding ciphertext, it remains difficult to find the key.

The specific technique makes repeated use of DES, iterating the encipherment $t = 25$ times (see Figure 10.2). In detail, a user password is truncated to its first 8 ASCII characters. Each of these provides 7 bits for a 56-bit DES key (padded with 0-bits if less than 8 characters). The key is used to DES-encrypt the 64-bit constant 0, with the output fed back as input t times iteratively. The 64-bit result is repacked into 11 printable characters (a 64-bit output and 12 salt bits yields 76 bits; 11 ASCII characters allow 77). In addition, a non-standard method of password salting is used, intended to simultaneously complicate dictionary attacks and preclude use of off-the-shelf DES hardware for attacks:

1. *password salting*. UNIX password salting associates a 12-bit “random” salt (12 bits taken from the system clock at time of password creation) with each user-selected password. The 12 bits are used to alter the standard expansion function E of the DES mapping (see §7.4), providing one of 4096 variations. (The expansion E creates a 48-bit block; immediately thereafter, the salt bits collectively determine one of 4096 permutations. Each bit is associated with a pre-determined pair from the 48-bit block, e.g., bit 1 with block bits 1 and 25, bit 2 with block bits 2 and 26, etc. If the salt bit is 1, the block bits are swapped, and otherwise they are not.) Both the hashed password and salt are recorded in the system password file. Security of any particular user’s password is unchanged by salting, but a dictionary attack now requires $2^{12} = 4096$ variations of each trial password.
2. *preventing use of off-the-shelf DES chips*. Because the DES expansion permutation E is dependent on the salt, standard DES chips can no longer be used to implement the UNIX password algorithm. An adversary wishing to use hardware to speed up an attack must build customized hardware rather than use commercially available chips. This may deter adversaries with modest resources.

The value stored for a given userid in the write-protected password file `/etc/passwd` is thus the iterated encryption of 0 under that user’s password, using the salted modification of DES. The constant 0 here could be replaced by other values, but typically is not. The overall algorithm is called the UNIX *crypt* password algorithm.

²UNIX is a trademark of Bell Laboratories.

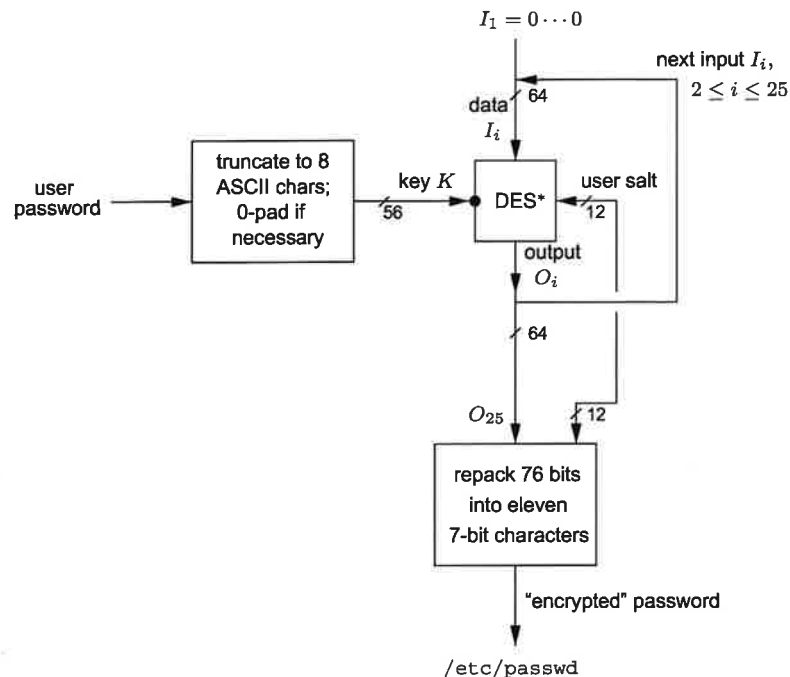


Figure 10.2: UNIX crypt password mapping. DES^* indicates DES with the expansion mapping E modified by a 12-bit salt.

10.5 Remark (performance advances) While the UNIX *crypt* mapping with $t = 25$ iterations provided a reasonable measure of protection against exhaustive search when introduced in the 1970s, for equivalent security in a system designed today a more computationally intensive mapping would be provided, due to performance advances in both hardware and software.

10.2.4 PINs and passkeys

(i) PINs

Personal identification numbers (PINs) fall under the category of fixed (time-invariant) passwords. They are most often used in conjunction with “something possessed”, typically a physical *token* such as a plastic banking card with a magnetic stripe, or a chipcard. To prove one’s identity as the authorized user of the token, and gain access to the privileges associated therewith, entry of the correct PIN is required when the token is used. This provides a second level of security if the token is lost or stolen. PINs may also serve as the second level of security for entry to buildings which have an independent first level of security (e.g., a security guard or video camera).

For user convenience and historical reasons, PINs are typically short (relative to fixed password schemes) and numeric, e.g., 4 to 8 digits. To prevent exhaustive search through such a small key space (e.g., 10 000 values for a 4-digit numeric PIN), additional procedural constraints are necessary. For example, some automated cash dispenser machines accessed

by banking cards confiscate a card if three incorrect PINs are entered successively; for others, incorrect entry of a number of successive PINs may cause the card to be “locked” or deactivated, thereafter requiring a longer PIN (e.g., 8 digits) for reactivation following such suspicious circumstances.

In an on-line system using PINs or reusable passwords, a claimed identity accompanied by a user-entered PIN may be verified by comparison to the PIN stored for that identity in a system database. An alternative is to use the PIN as a key for a MAC (see Chapter 9).

In an off-line system without access to a central database, information facilitating PIN verification must be stored on the token itself. If the PIN need not be user-selected, this may be done by defining the PIN to be a function of a secret key and the identity associated with the token; the PIN is then verifiable by any remote system knowing this master key.

In an off-line system, it may also be desirable to allow the PIN to be user-selectable, to facilitate PIN memorization by users. In this case, the PIN may be encrypted under a master key and stored on the token, with the master key known to all off-line terminals that need to be capable of verifying the token. A preferable design is to store a one-way function of the PIN, user identity, and master key on the token.

(ii) Two-stage authentication and password-derived keys

Human users have difficulty remembering secret keys which have sufficient entropy to provide adequate security. Two techniques which address this issue are now described.

When tokens are used with off-line PIN verification, a common technique is for the PIN to serve to verify the user to the token, while the token contains additional independent information allowing the token to authenticate itself to the system (as a valid token representing a legitimate user). The user is thereby indirectly authenticated to the system by a two-stage process. This requires the user have possession of the token but need remember only a short PIN, while a longer key (containing adequate entropy) provides cryptographic security for authentication over an unsecured link.

A second technique is for a user password to be mapped by a one-way hash function into a cryptographic key (e.g., a 56-bit DES key). Such password-derived keys are called *passkeys*. The passkey is then used to secure a communications link between the user and a system which also knows the user password. It should be ensured that the entropy of the user’s password is sufficiently large that exhaustive search of the password space is not more efficient than exhaustive search of the passkey space (i.e., guessing passwords is not easier than guessing 56-bit DES keys); see Table 10.1 for guidance.

An alternative to having passkeys remain fixed until the password is changed is to keep a running sequence number on the system side along with each user’s password, for use as a time-variant salt communicated to the user in the clear and incremented after each use. A fixed per-user salt could also be used in addition to a running sequence number.

Passkeys should be viewed as long-term keys, with use restricted to authentication and key management (e.g., rather than also for bulk encryption of user data). A disadvantage of using password-derived keys is that storing each user’s password within the system requires some mechanism to protect the confidentiality of the stored passwords.

10.2.5 One-time passwords (towards strong authentication)

A natural progression from fixed password schemes to challenge-response identification protocols may be observed by considering one-time password schemes. As was noted in §10.2.2, a major security concern of fixed password schemes is eavesdropping and subsequent replay of the password. A partial solution is *one-time passwords*: each password is

used only once. Such schemes are safe from passive adversaries who eavesdrop and later attempt impersonation. Variations include:

1. *shared lists of one-time passwords*. The user and the system use a sequence or set of t secret passwords, (each valid for a single authentication), distributed as a pre-shared list. A drawback is maintenance of the shared list. If the list is not used sequentially, the system may check the entered password against all remaining unused passwords. A variation involves use of a *challenge-response table*, whereby the user and the system share a table of matching challenge-response pairs, ideally with each pair valid at most once; this non-cryptographic technique differs from the cryptographic challenge-response of §10.3.
2. *sequentially updated one-time passwords*. Initially only a single secret password is shared. During authentication using password i , the user creates and transmits to the system a new password (password $i + 1$) encrypted under a key derived from password i . This method becomes difficult if communication failures occur.
3. *one-time password sequences based on a one-way function*. Lamport's one-time password scheme is described below. This method is more efficient (with respect to bandwidth) than sequentially updated one-time passwords, and may be viewed as a challenge-response protocol where the challenge is implicitly defined by the current position within the password sequence.

One-time passwords based on one-way functions (Lamport's scheme)

In Lamport's one-time password scheme, the user begins with a secret w . A one-way function (OWF) H is used to define the password sequence: $w, H(w), H(H(w)), \dots, H^t(w)$. The password for the i^{th} identification session, $1 \leq i \leq t$, is defined to be $w_i = H^{t-i}(w)$.

10.6 Protocol Lamport's OWF-based one-time passwords

SUMMARY: A identifies itself to B using one-time passwords from a sequence.

1. *One-time setup*.
 - (a) User A begins with a secret w . Let H be a one-way function.
 - (b) A constant t is fixed (e.g., $t = 100$ or 1000), defining the number of identifications to be allowed. (The system is thereafter restarted with a new w , to avoid replay attacks.)
 - (c) A transfers (the *initial shared secret*) $w_0 = H^t(w)$, in a manner guaranteeing its authenticity, to the system B . B initializes its counter for A to $i_A = 1$.
2. *Protocol messages*. The i^{th} identification, $1 \leq i \leq t$, proceeds as follows:

$$A \rightarrow B : A, i, w_i (= H^{t-i}(w)) \quad (1)$$

Here $A \rightarrow B : X$ denotes A sending the message X to B .

3. *Protocol actions*. To identify itself for session i , A does the following.
 - (a) A 's equipment computes $w_i = H^{t-i}(w)$ (easily done either from w itself, or from an appropriate intermediate value saved during the computation of $H^t(w)$ initially), and transmits (1) to B .
 - (b) B checks that $i = i_A$, and that the received password w_i satisfies: $H(w_i) = w_{i-1}$. If both checks succeed, B accepts the password, sets $i_A \leftarrow i_A + 1$, and saves w_i for the next session verification.

10.7 Note (*pre-play attack*) Protocol 10.6 and similar one-time password schemes including that of Note 10.8 remain vulnerable to an active adversary who intercepts and traps (or impersonates the system in order to extract) an as-yet unused one-time password, for the purpose of subsequent impersonation. To prevent this, a password should be revealed only to a party which itself is known to be authentic. Challenge-response techniques (see §10.3) address this threat.

10.8 Note (*alternative one-time password scheme*) The following one-time-password alternative to Protocol 10.6 is suitable if storing actual passwords on the system side is acceptable (cf. Figure 10.1; compare also to §10.3.2(iii)). The claimant A has a shared password P with the system verifier B , to which it sends the data pair: $(r, H(r, P))$. The verifier computes the hash of the received value r and its local copy of P , and declares acceptance if this matches the received hash value. To avoid replay, r should be a sequence number, timestamp, or other parameter which can be easily guaranteed to be accepted only once.

10.3 Challenge-response identification (strong authentication)

The idea of cryptographic challenge-response protocols is that one entity (the claimant) “proves” its identity to another entity (the verifier) by demonstrating knowledge of a secret known to be associated with that entity, without revealing the secret itself to the verifier during the protocol.³ This is done by providing a response to a time-variant challenge, where the response depends on both the entity’s secret and the challenge. The *challenge* is typically a number chosen by one entity (randomly and secretly) at the outset of the protocol. If the communications line is monitored, the response from one execution of the identification protocol should not provide an adversary with useful information for a subsequent identification, as subsequent challenges will differ.

Before considering challenge-response identification protocols based on symmetric-key techniques (§10.3.2), public-key techniques (§10.3.3), and zero-knowledge concepts (§10.4), background on time-variant parameters is first provided.

10.3.1 Background on time-variant parameters

Time-variant parameters may be used in identification protocols to counteract replay and interleaving attacks (see §10.5), to provide uniqueness or timeliness guarantees, and to prevent certain chosen-text attacks. They may similarly be used in authenticated key establishment protocols (Chapter 12), and to provide uniqueness guarantees in conjunction with message authentication (Chapter 9).

Time-variant parameters which serve to distinguish one protocol instance from another are sometimes called *nonces*, *unique numbers*, or *non-repeating values*; definitions of these terms have traditionally been loose, as the specific properties required depend on the actual usage and protocol.

10.9 Definition A *nonce* is a value used no more than once for the same purpose. It typically serves to prevent (undetectable) replay.

³In some mechanisms, the secret is known to the verifier, and is used to verify the response; in others, the secret need not actually be known by the verifier.

The term *nonce* is most often used to refer to a “random” number in a challenge-response protocol, but the required randomness properties vary. Three main classes of time-variant parameters are discussed in turn below: random numbers, sequence numbers, and time-stamps. Often, to ensure protocol security, the integrity of such parameters must be guaranteed (e.g., by cryptographically binding them with other data in a challenge-response sequence). This is particularly true of protocols in which the only requirement of a time-variant parameter is uniqueness, e.g., as provided by a never-repeated sequential counter.⁴

Following are some miscellaneous points about time-variant parameters.

1. Verifiable timeliness may be provided through use of random numbers in challenge-response mechanisms, timestamps in conjunction with distributed timeclocks, or sequence numbers in conjunction with the maintenance of pairwise (claimant, verifier) state information.
2. To provide timeliness or uniqueness guarantees, the verifier in the protocol controls the time-variant parameter, either directly (through choice of a random number) or indirectly (through information maintained regarding a shared sequence, or logically through a common time clock).
3. To uniquely identify a message or sequence of messages (protocol instance), nonces drawn from a monotonically increasing sequence may be used (e.g., sequence or serial numbers, and timestamps, if guaranteed to be increasing and unique), or random numbers of sufficient size. Uniqueness is often required only within a given key lifetime or time window.
4. Combinations of time-variant parameters may be used, e.g., random numbers concatenated to timestamps or sequence numbers. This may guarantee that a pseudorandom number is not duplicated.

(i) Random numbers

Random numbers may be used in challenge-response mechanisms, to provide uniqueness and timeliness assurances, and to preclude certain replay and interleaving attacks (see §10.5, including Remark 10.42). Random numbers may also serve to provide unpredictability, for example, to preclude chosen-text attacks.

The term *random numbers*, when used in the context of identification and authentication protocols, includes pseudorandom numbers which are unpredictable to an adversary (see Remark 10.11); this differs from randomness in the traditional statistical sense. In protocol descriptions, “choose a random number” is usually intended to mean “pick a number with uniform distribution from a specified sample space” or “select from a uniform distribution”.

Random numbers are used in challenge-response protocols as follows. One entity includes a (new) random number in an outgoing message. An incoming message subsequently received (e.g., the next protocol message of the same protocol instance), whose construction required knowledge of this nonce and to which this nonce is inseparably bound, is then deemed to be *fresh* (Remark 10.10) based on the reasoning that the random number links the two messages. The non-tamperable binding is required to prevent appending a nonce to an old message.

Random numbers used in this manner serve to fix a relative point in time for the parties involved, analogous to a shared timeclock. The maximum allowable time between protocol messages is typically constrained by a *timeout period*, enforced using local, independent countdown timers.

⁴Such predictable parameters differ from sequence numbers in that they might not be bound to any stored state. Without appropriate cryptographic binding, a potential concern then is a pre-play attack wherein an adversary obtains the response before the time-variant parameter is legitimately sent (see Note 10.7).

- 10.10 Remark** (*freshness*) In the context of challenge-response protocols, *fresh* typically means recent, in the sense of having originated subsequent to the beginning of the current protocol instance. Note that such freshness alone does not rule out interleaving attacks using parallel sessions (see §10.5).
- 10.11 Remark** (*birthday repetitions in random numbers*) In generating pseudorandom numbers for use as time-variant parameters, it suffices if the probability of a repeated number is acceptably low and if numbers are not intentionally reused. This may be achieved by selecting the random value from a sufficiently large sample space, taking into account coincidences arising from the birthday paradox. The latter may be addressed by either using a larger sample space, or by using a generation process guaranteed to avoid repetition (e.g., a bijection), such as using the counter or OFB mode of a block cipher (§7.2.2).
- 10.12 Remark** (*disadvantages of random numbers*) Many protocols involving random numbers require the generation of cryptographically secure (i.e., unpredictable) random numbers. If pseudorandom number generators are used, an initial seed with sufficient entropy is required. When random numbers are used in challenge-response mechanisms in place of timestamps, typically the protocol involves one additional message, and the challenger must temporarily maintain state information, but only until the response is verified.

(ii) Sequence numbers

A sequence number (serial number, or counter value) serves as a unique number identifying a message, and is typically used to detect message replay. For stored files, sequence numbers may serve as *version numbers* for the file in question. Sequence numbers are specific to a particular pair of entities, and must explicitly or implicitly be associated with both the originator and recipient of a message; distinct sequences are customarily necessary for messages from *A* to *B* and from *B* to *A*.

Parties follow a pre-defined policy for message numbering. A message is accepted only if the sequence number therein has not been used previously (or not used previously within a specified time period), and satisfies the agreed policy. The simplest policy is that a sequence number starts at zero, is incremented sequentially, and each successive message has a number one greater than the previous one received. A less restrictive policy is that sequence numbers need (only) be monotonically increasing; this allows for lost messages due to non-malicious communications errors, but precludes detection of messages lost due to adversarial intervention.

- 10.13 Remark** (*disadvantages of sequence numbers*) Use of sequence numbers requires an overhead as follows: each claimant must record and maintain long-term pairwise state information for each possible verifier, sufficient to determine previously used and/or still valid sequence numbers. Special procedures (e.g., for resetting sequence numbers) may be necessary following circumstances disrupting normal sequencing (e.g., system failures). Forced delays are not detectable in general. As a consequence of the overhead and synchronization necessary, sequence numbers are most appropriate for smaller, closed groups.

(iii) Timestamps

Timestamps may be used to provide timeliness and uniqueness guarantees, to detect message replay. They may also be used to implement time-limited access privileges, and to detect forced delays.

Timestamps function as follows. The party originating a message obtains a timestamp from its local (host) clock, and cryptographically binds it to a message. Upon receiving a time-stamped message, the second party obtains the current time from its own (host) clock, and subtracts the timestamp received. The received message is valid provided:

1. the timestamp difference is within the *acceptance window* (a fixed-size time interval, e.g., 10 milliseconds or 20 seconds, selected to account for the maximum message transit and processing time, plus clock skew); and
2. (optionally) no message with an identical timestamp has been previously received from the same originator. This check may be made by the verifier maintaining a list of all timestamps received from each source entity within the current acceptance window. Another method is to record the latest (valid) timestamp used by each source (in this case the verifier accepts only strictly increasing time values).

The security of timestamp-based verification relies on use of a common time reference. This requires that host clocks be available and both “loosely synchronized” and secured from modification. Synchronization is necessary to counter clock drift, and must be appropriate to accommodate the acceptance window used. The degree of clock skew allowed, and the acceptance window, must be appropriately small to preclude message replay if the above optional check is omitted. The timeclock must be secure to prevent adversarial resetting of a clock backwards so as to restore the validity of old messages, or setting a clock forward to prepare a message for some future point in time (cf. Note 10.7).

10.14 Remark (*disadvantages of timestamps*) Timestamp-based protocols require that timeclocks be both synchronized and secured. The preclusion of adversarial modification of local timeclocks is difficult to guarantee in many distributed environments; in this case, the security provided must be carefully re-evaluated. Maintaining lists of used timestamps within the current window has the drawback of a potentially large storage requirement, and corresponding verification overhead. While technical solutions exist for synchronizing distributed clocks, if synchronization is accomplished via network protocols, such protocols themselves must be secure, which typically requires authentication; this leads to a circular security argument if such authentication is itself timestamp-based.

10.15 Remark (*comparison of time-variant parameters*) Timestamps in protocols offer the advantage of fewer messages (typically by one), and no requirement to maintain pairwise long-term state information (cf. sequence numbers) or per-connection short-term state information (cf. random numbers). Minimizing state information is particularly important for servers in client-server applications. The main drawback of timestamps is the requirement of maintaining secure, synchronized distributed timeclocks. Timestamps in protocols may typically be replaced by a random number challenge plus a return message.

10.3.2 Challenge-response by symmetric-key techniques

Challenge-response mechanisms based on symmetric-key techniques require the claimant and the verifier to share a symmetric key. For closed systems with a small number of users, each pair of users may share a key a priori; in larger systems employing symmetric-key techniques, identification protocols often involve the use of a trusted on-line server with which each party shares a key. The on-line server effectively acts like the hub of a spoked wheel, providing a common session key to two parties each time one requests authentication with the other.

The apparent simplicity of the techniques presented below and in §10.3.3 is misleading. The design of such techniques is intricate and the security is brittle; those presented have been carefully selected.

(i) Challenge-response based on symmetric-key encryption

Both the Kerberos protocol (Protocol 12.24) and the Needham-Schroeder shared-key protocol (Protocol 12.26) provide entity authentication based on symmetric encryption and involve use of an on-line trusted third party. These are discussed in Chapter 12, as they additionally provide key establishment.

Below, three simple techniques based on ISO/IEC 9798-2 are described. They assume the prior existence of a shared secret key (and no further requirement for an on-line server). In this case, two parties may carry out unilateral entity authentication in one pass using timestamps or sequence numbers, or two passes using random numbers; mutual authentication requires, respectively, two and three passes. The claimant corroborates its identity by demonstrating knowledge of the shared key by encrypting a challenge (and possibly additional data) using the key. These techniques are similar to those given in §12.3.1.

10.16 Remark (*data integrity*) When encipherment is used in entity authentication protocols, data integrity must typically also be guaranteed to ensure security. For example, for messages spanning more than one block, the rearrangement of ciphertext blocks cannot be detected in the ECB mode of block encryption, and even CBC encryption may provide only a partial solution. Such data integrity should be provided through use of an accepted data integrity mechanism (see §9.6; cf. Remark 12.19).

9798-2 mechanisms: Regarding notation: r_A and t_A , respectively, denote a random number and a timestamp, generated by A . (In these mechanisms, the timestamp t_A may be replaced by a sequence number n_A , providing slightly different guarantees.) E_K denotes a symmetric encryption algorithm, with a key K shared by A and B ; alternatively, distinct keys K_{AB} and K_{BA} may be used for unidirectional communication. It is assumed that both parties are aware of the claimed identity of the other, either by context or by additional (unsecured) cleartext data fields. Optional message fields are denoted by an asterisk (*), while a comma (,) within the scope of E_K denotes concatenation.

1. unilateral authentication, timestamp-based:

$$A \rightarrow B : E_K(t_A, B^*) \quad (1)$$

Upon reception and decryption, B verifies that the timestamp is acceptable, and optionally verifies the received identifier as its own. The identifier B here prevents an adversary from re-using the message immediately on A , in the case that a single bi-directional key K is used.

2. unilateral authentication, using random numbers:

To avoid reliance on timestamps, the timestamp may be replaced by a random number, at the cost of an additional message:

$$A \leftarrow B : r_B \quad (1)$$

$$A \rightarrow B : E_K(r_B, B^*) \quad (2)$$

B decrypts the received message and checks that the random number matches that sent in (1). Optionally, B checks that the identifier in (2) is its own; this prevents a reflection attack in the case of a bi-directional key K . To prevent chosen-text attacks on the encryption scheme E_K , A may (as below) embed an additional random number in (2) or, alternately, the form of the challenges can be restricted; the critical requirement is that they be non-repeating.

Handbook of Applied Cryptography by A. Menezes, P. van Oorschot and S. Vanstone.

3. *mutual authentication, using random numbers:*

$$A \leftarrow B : r_B \quad (1)$$

$$A \rightarrow B : E_K(r_A, r_B, B^*) \quad (2)$$

$$A \leftarrow B : E_K(r_B, r_A) \quad (3)$$

Upon reception of (2), B carries out the checks as above and, in addition, recovers the decrypted r_A for inclusion in (3). Upon decrypting (3), A checks that both random numbers match those used earlier. The second random number r_A in (2) serves both as a challenge and to prevent chosen-text attacks.

10.17 Remark (*doubling unilateral authentication*) While mutual authentication may be obtained by running any of the above unilateral authentication mechanisms twice (once in each direction), such an ad-hoc combination suffers the drawback that the two unilateral authentications, not being linked, cannot logically be associated with a single protocol run.

(ii) Challenge-response based on (keyed) one-way functions

The encryption algorithm in the above mechanisms may be replaced by a one-way or non-reversible function of the shared key and challenge, e.g., having properties similar to a MAC (Definition 9.7). This may be preferable in situations where encryption algorithms are otherwise unavailable or undesirable (e.g., due to export restrictions or computational costs). The modifications required to the 9798-2 mechanisms above (yielding the analogous mechanisms of ISO/IEC 9798-4) are the following:

1. the encryption function E_K is replaced by a MAC algorithm h_K ;
2. rather than decrypting and verifying that fields match, the recipient now independently computes the MAC value from known quantities, and accepts if the computed MAC matches the received MAC value; and
3. to enable independent MAC computation by the recipient, the additional cleartext field t_A must be sent in message (1) of the one-pass mechanism. r_A must be sent as an additional cleartext field in message (2) of the three-pass mechanism.

The revised three-pass challenge-response mechanism based on a MAC h_K , with actions as noted above, provides mutual identification. Essentially the same protocol, called *SKID3*, has messages as follows:

$$A \leftarrow B : r_B \quad (1)$$

$$A \rightarrow B : r_A, h_K(r_A, r_B, B) \quad (2)$$

$$A \leftarrow B : h_K(r_B, r_A, A) \quad (3)$$

Note that the additional field A is included in message (3). The protocol *SKID2*, obtained by omitting the third message, provides unilateral entity authentication.

(iii) Implementation using hand-held passcode generators

Answering a challenge in challenge-response protocols requires some type of computing device and secure storage for long-term keying material (e.g., a file on a trusted local disk, perhaps secured under a local password-derived key). For additional security, a device such as a chipcard (and corresponding card reader) may be used for both the key storage and response computation. In some cases, a less expensive option is a passcode generator.

Passcode generators are hand-held devices, resembling thin calculators in both size and display, and which provide time-variant passwords or *passcodes* (see Figure 10.3). The generator contains a device-specific secret key. When a user is presented with a challenge (e.g., by a system displaying it on a computer terminal), the challenge is keyed into the generator. The generator displays a passcode, computed as a function of the secret key and the

challenge; this may be either an asymmetric function, or a symmetric function (e.g., encryption or MAC as discussed above). The user returns the response (e.g., keys the passcode in at his terminal), which the system verifies by comparison to an independently computed response, using the same information stored on the system side.

For further protection against misplaced generators, the response may also depend on a user-entered PIN. Simpler passcode generators omit the user keypad, and use as an implicit challenge a time value (with a typical granularity of one minute) defined by a timeclock loosely synchronized automatically between the system and the passcode generator. A more sophisticated device combines implicit synchronization with explicit challenges, presenting an explicit challenge only when synchronization is lost.

A drawback of systems using passcode generators is, as per §10.2.1(i), the requirement to provide confidentiality for user passwords stored on the system side.

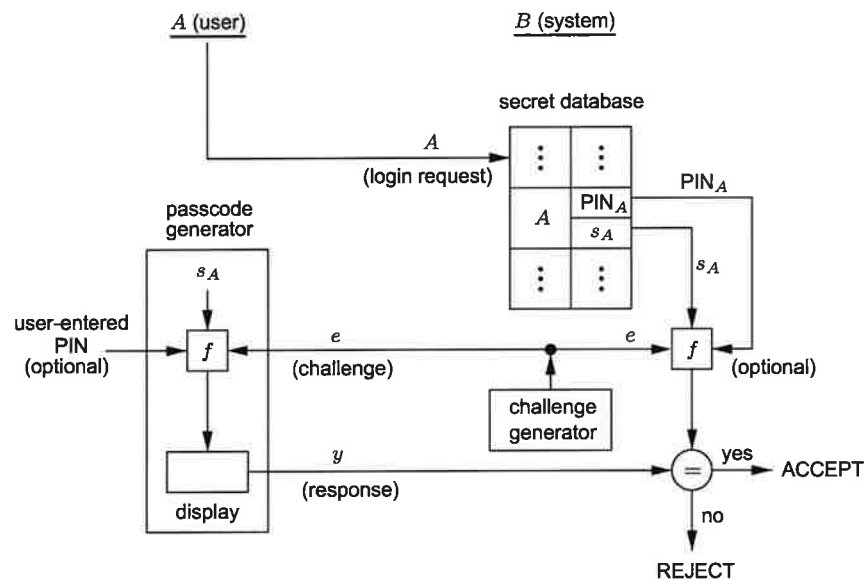


Figure 10.3: Functional diagram of a hand-held passcode generator. s_A is A's user-specific secret. f is a one-way function. The (optional) PIN could alternatively be locally verified in the passcode generator only, making y independent of it.

10.3.3 Challenge-response by public-key techniques

Public-key techniques may be used for challenge-response based identification, with a claimant demonstrating knowledge of its private key in one of two ways (cf. §12.5):

1. the claimant decrypts a challenge encrypted under its public key;
2. the claimant digitally signs a challenge.

Ideally, the public-key pair used in such mechanisms should not be used for other purposes, since combined usage may compromise security (Remark 10.40). A second caution is that the public-key system used should not be susceptible to chosen-ciphertext attacks,⁵

⁵Both chosen-ciphertext and chosen-plaintext attacks are of concern for challenge-response techniques based on symmetric-key encryption.

as an adversary may attempt to extract information by impersonating a verifier and choosing strategic rather than random challenges. (See Notes 8.13 and 8.58 regarding the Rabin/Williams and Blum-Goldwasser schemes.)

Incorporating a self-generated random number or *confounder* (§10.5) into the data over which the response is computed may address both of these concerns. Such data may be made available to the verifier in cleartext to allow verification.

(i) Challenge-response based on public-key decryption

Identification based on PK decryption and witness. Consider the following protocol:

$$A \leftarrow B : h(r), B, P_A(r, B) \quad (1)$$

$$A \rightarrow B : r \quad (2)$$

B chooses a random r , computes the *witness* $x = h(r)$ (x demonstrates knowledge of r without disclosing it – cf. §10.4.1), and computes the challenge $e = P_A(r, B)$. Here P_A denotes the public-key encryption (e.g., RSA) algorithm of A , and h denotes a one-way hash function. B sends (1) to A . A decrypts e to recover r' and B' , computes $x' = h(r')$, and quits if $x' \neq x$ (implying $r' \neq r$) or if B' is not equal to its own identifier B . Otherwise, A sends $r = r'$ to B . B succeeds with (unilateral) entity authentication of A upon verifying the received r agrees with that sent earlier. The use of the witness precludes chosen-text attacks.

Modified Needham-Schroeder PK protocol for identification. The modified Needham-Schroeder public-key protocol of Note 12.39 provides key transport of distinct keys k_1, k_2 from A to B and B to A , respectively, as well as mutual authentication. If the key establishment feature is not required, k_1 and k_2 may be omitted. With P_B denoting the public-key encryption algorithm for B (e.g., RSA), the messages in the modified protocol for identification are then as follows:

$$A \rightarrow B : P_B(r_1, A) \quad (1)$$

$$A \leftarrow B : P_A(r_1, r_2) \quad (2)$$

$$A \rightarrow B : r_2 \quad (3)$$

Verification actions are analogous to those of Note 12.39.

(ii) Challenge-response based on digital signatures

X.509 mechanisms based on digital signatures. The ITU-T (formerly CCITT) X.509 two- and three-way strong authentication protocols specify identification techniques based on digital signatures and, respectively, timestamps and random number challenges. These are described in §12.5.2, and optionally provide key establishment in addition to entity authentication.

9798-3 mechanisms. Three challenge-response identification mechanisms based on signatures are given below, analogous to those in §10.3.2(i) based on symmetric-key encryption, but, in this case, corresponding to techniques in ISO/IEC 9798-3. Regarding notation (cf. 9798-2 above): r_A and t_A , respectively, denote a random number and timestamp generated by A . S_A denotes A 's signature mechanism; if this mechanism provides message recovery, some of the cleartext fields listed below are redundant and may be omitted. $cert_A$ denotes the public-key certificate containing A 's signature public key. (In these mechanisms, if the verifier has the authentic public key of the claimant a priori, certificates may be omitted; otherwise, it is assumed that the verifier has appropriate information to verify the validity of the public key contained in a received certificate – see Chapter 13.) Remark 10.17 also applies here.

©1997 by CRC Press, Inc. — See accompanying notice at front of chapter.

1. *unilateral authentication with timestamps:*

$$A \rightarrow B : cert_A, t_A, B, S_A(t_A, B) \quad (1)$$

Upon reception, B verifies that the timestamp is acceptable, the received identifier B is its own, and (using A 's public key extracted from $cert_A$ after verifying the latter) checks that the signature over these two fields is correct.

2. *unilateral authentication with random numbers:* Reliance on timestamps may be replaced by a random number, at the cost of an additional message:

$$A \leftarrow B : r_B \quad (1)$$

$$A \rightarrow B : cert_A, r_A, B, S_A(r_A, r_B, B) \quad (2)$$

B verifies that the cleartext identifier is its own, and using a valid signature public key for A (e.g., from $cert_A$), verifies that A 's signature is valid over the cleartext random number r_A , the same number r_B as sent in (1), and this identifier. The signed r_A explicitly prevents chosen-text attacks.

3. *mutual authentication with random numbers:*

$$A \leftarrow B : r_B \quad (1)$$

$$A \rightarrow B : cert_A, r_A, B, S_A(r_A, r_B, B) \quad (2)$$

$$A \leftarrow B : cert_B, A, S_B(r_B, r_A, A) \quad (3)$$

Processing of (1) and (2) is as above; (3) is processed analogously to (2).

10.4 Customized and zero-knowledge identification protocols

This section considers protocols specifically designed to achieve identification, which use asymmetric techniques but do not rely on digital signatures or public-key encryption, and which avoid use of block ciphers, sequence numbers, and timestamps. They are similar in some regards to the challenge-response protocols of §10.3, but are based on the ideas of interactive proof systems and zero-knowledge proofs (see §10.4.1), employing random numbers not only as challenges, but also as *commitments* to prevent cheating.

10.4.1 Overview of zero-knowledge concepts

A disadvantage of simple password protocols is that when a claimant A (called a *prover* in the context of zero-knowledge protocols) gives the verifier B her password, B can thereafter impersonate A . Challenge-response protocols improve on this: A responds to B 's challenge to demonstrate knowledge of A 's secret in a time-variant manner, providing information not directly reusable by B . This might nonetheless reveal some partial information about the claimant's secret; an adversarial verifier might also be able to strategically select challenges to obtain responses providing such information (see chosen-text attacks, §10.5).

Zero-knowledge (ZK) protocols are designed to address these concerns, by allowing a prover to demonstrate knowledge of a secret while revealing no information whatsoever (beyond what the verifier was able to deduce prior to the protocol run) of use to the verifier

Handbook of Applied Cryptography by A. Menezes, P. van Oorschot and S. Vanstone.

in conveying this demonstration of knowledge to others. The point is that only a single bit of information need be conveyed – namely, that the prover actually does know the secret.

More generally, a zero-knowledge protocol allows a proof of the truth of an assertion, while conveying no information whatsoever (this notion can be quantified in a rigorous sense) about the assertion itself other than its actual truth. In this sense, a zero-knowledge proof is similar to an answer obtained from a (trusted) *oracle*.

(i) Interactive proof systems and zero-knowledge protocols

The ZK protocols to be discussed are instances of *interactive proof systems*, wherein a prover and verifier exchange multiple messages (challenges and responses), typically dependent on random numbers (ideally: the outcomes of fair coin tosses) which they may keep secret. The prover's objective is to convince (*prove to*) the verifier the truth of an assertion, e.g., claimed knowledge of a secret. The verifier either accepts or rejects the *proof*. The traditional mathematical notion of a proof, however, is altered to an interactive game wherein proofs are *probabilistic* rather than absolute; a proof in this context need be correct only with bounded probability, albeit possibly arbitrarily close to 1. For this reason, an interactive proof is sometimes called a *proof by protocol*.

Interactive proofs used for identification may be formulated as proofs of knowledge. *A* possesses some secret *s*, and attempts to convince *B* it has *knowledge* of *s* by correctly responding to queries (involving publicly known inputs and agreed upon functions) which require knowledge of *s* to answer. Note that proving knowledge of *s* differs from proving that such *s* exists – for example, proving knowledge of the prime factors of *n* differs from proving that *n* is composite.

An interactive proof is said to be a *proof of knowledge* if it has both the properties of completeness and soundness. Completeness may be viewed as the customary requirement that a protocol functions properly given honest participants.

10.18 Definition (*completeness property*) An interactive proof (protocol) is *complete* if, given an honest prover and an honest verifier, the protocol succeeds with overwhelming probability (i.e., the verifier accepts the prover's claim). The definition of *overwhelming* depends on the application, but generally implies that the probability of failure is not of practical significance.

10.19 Definition (*soundness property*) An interactive proof (protocol) is *sound* if there exists an expected polynomial-time algorithm *M* with the following property: if a dishonest prover (impersonating *A*) can with non-negligible probability successfully execute the protocol with *B*, then *M* can be used to extract from this prover knowledge (essentially equivalent to *A*'s secret) which with overwhelming probability allows successful subsequent protocol executions.

An alternate explanation of the condition in Definition 10.19 is as follows: the prover's secret *s* together with public data satisfies some polynomial-time predicate, and another solution of this predicate (possibly the same) can be extracted, allowing successful execution of subsequent protocol instances.

Since any party capable of impersonating *A* must know the equivalent of *A*'s secret knowledge (*M* can be used to extract it from this party in polynomial time), soundness guarantees that the protocol does indeed provide a proof of knowledge – knowledge equivalent to that being queried is required to succeed. Soundness thus prevents a dishonest prover from convincing an honest verifier (but does not by itself guarantee that acquiring the

prover's secret is difficult; see Remark 10.23). A standard method to establish the soundness of a particular protocol is to assume the existence of a dishonest prover capable of successfully executing the protocol, and show how this allows one to compute the real prover's secret.

While an interactive proof of knowledge (or protocol based thereon) must be sound to be of cryptographic use, the main property of zero-knowledge protocols is the zero-knowledge aspect itself. For what follows, define a *transcript* (or view) to be the collection of messages resulting from protocol execution.

10.20 Definition (*zero-knowledge property*) A protocol which is a proof of knowledge has the *zero-knowledge property* if it is simulatable in the following sense: there exists an expected polynomial-time algorithm (*simulator*) which can produce, upon input of the assertion(s) to be proven but without interacting with the real prover, transcripts indistinguishable from those resulting from interaction with the real prover.

The zero-knowledge property implies that a prover executing the protocol (even when interacting with a malicious verifier) does not release any information (about its secret knowledge, other than that the particular assertion itself is true) not otherwise computable in polynomial time from public information alone. Thus, participation does not increase the chances of subsequent impersonation.

10.21 Remark (*simulated ZK protocols and protocol observers*) Consider an observer C who witnesses a zero-knowledge interactive proof (ZKIP) involving a prover A convincing a verifier B ($B \neq C$) of some knowledge A has. The “proof” to B does not provide any guarantees to C . (Indeed, A and B might have a prior agreement, conspiring against C , on the challenges to be issued.) Similarly, a recorded ZKIP conveys no guarantees upon playback. This is fundamental to the idea of the zero-knowledge property and the condition that proofs be simulatable by a verifier alone. Interactive proofs convey knowledge only to (interactive) verifiers able to select their own random challenges.

10.22 Definition (*computational vs. perfect zero-knowledge*) A protocol is *computationally* zero-knowledge if an observer restricted to probabilistic polynomial-time tests cannot distinguish real from simulated transcripts. For *perfect* zero-knowledge, the probability distributions of the transcripts must be identical. By convention, when not further qualified, *zero-knowledge* means computational zero-knowledge.

In the case of computational zero-knowledge, real and simulated transcripts are said to be *polynomially indistinguishable* (indistinguishable using polynomial-time algorithms). Any information extracted by a verifier through interaction with a prover provides no advantage to the verifier within polynomial time.

10.23 Remark (*ZK property and soundness vs. security*) The zero-knowledge property (Definition 10.20) does not guarantee that a protocol is secure (i.e., that the probability of it being easily defeated is negligible). Similarly, the soundness property (Definition 10.19) does not guarantee that a protocol is secure. Neither property has much value unless the underlying problem faced by an adversary is computationally hard.

(ii) Comments on zero-knowledge vs. other asymmetric protocols

The following observations may be made regarding zero-knowledge (ZK) techniques, as compared with other public-key (PK) techniques.

Handbook of Applied Cryptography by A. Menezes, P. van Oorschot and S. Vanstone.

1. *no degradation with usage*: protocols proven to have the ZK property do not suffer degradation of security with repeated use, and resist chosen-text attacks. This is perhaps the most appealing practical feature of ZK techniques.
A ZK technique which is not provably secure may or may not be viewed as more desirable than a PK technique which is provably secure (e.g., as difficult as factoring).
2. *encryption avoided*: many ZK techniques avoid use of explicit encryption algorithms. This may offer political advantages (e.g., with respect to export controls).
3. *efficiency*: while some ZK-based techniques are extremely efficient (see §10.4.5), protocols which formally have the zero-knowledge property typically have higher communications and/or computational overheads than PK protocols which do not. The computational efficiency of the more practical ZK-based schemes arises from their nature as interactive proofs, rather than their zero-knowledge aspect.
4. *unproven assumptions*: many ZK protocols (“proofs of knowledge”) themselves rely on the same unproven assumptions as PK techniques (e.g., the intractability of factoring or quadratic residuosity).
5. *ZK-based vs. ZK*: although supported by prudent underlying principles, many techniques based on zero-knowledge concepts fall short of formally being zero-knowledge and/or formally sound in practice, due to parameter selection for reasons of efficiency, or for other technical reasons (cf. Notes 10.33 and 10.38). In fact, many such concepts are asymptotic, and do not apply directly to practical protocols (Remark 10.34).

(iii) Example of zero-knowledge proof: Fiat-Shamir identification protocol

The general idea of a zero-knowledge (ZK) proof is illustrated by the basic version of the Fiat-Shamir protocol. The basic version is presented here for historical and illustrative purposes (Protocol 10.24). In practice, one would use a more efficient variation, such as Protocol 10.26, with multiple “questions” per iteration rather than as here, where B poses only a single one-bit challenge per iteration.

The objective is for A to identify itself by proving knowledge of a secret s (associated with A through authentic public data) to any verifier B , without revealing any information about s not known or computable by B prior to execution of the protocol (see Note 10.25). The security relies on the difficulty of extracting square roots modulo large composite integers n of unknown factorization, which is equivalent to that of factoring n (Fact 3.46).

10.24 Protocol Fiat-Shamir identification protocol (basic version)

SUMMARY: A proves knowledge of s to B in t executions of a 3-pass protocol.

1. *One-time setup*.
 - (a) A trusted center T selects and publishes an RSA-like modulus $n = pq$ but keeps primes p and q secret.
 - (b) Each claimant A selects a secret s coprime to n , $1 \leq s \leq n - 1$, computes $v = s^2 \bmod n$, and registers v with T as its public key.⁶
2. *Protocol messages*. Each of t rounds has three messages with form as follows.

$$A \rightarrow B : x = r^2 \bmod n \quad (1)$$

$$A \leftarrow B : e \in \{0, 1\} \quad (2)$$

$$A \rightarrow B : y = r \cdot s^e \bmod n \quad (3)$$

⁶Technically, T should verify the condition $\gcd(s, n) = 1$ or equivalently $\gcd(v, n) = 1$, for this to be a sound proof of knowledge; and B should stop with failure if $\gcd(y, n) \neq 1$, where y is A 's response in the third message. But either condition failing would allow the factorization of n , violating the assumption that n cannot be factored.

3. *Protocol actions.* The following steps are iterated t times (sequentially and independently). B accepts the proof if all t rounds succeed.

- (a) A chooses a random (*commitment*) r , $1 \leq r \leq n-1$, and sends (the *witness*) $x = r^2 \bmod n$ to B .
- (b) B randomly selects a (*challenge*) bit $e = 0$ or $e = 1$, and sends e to A .
- (c) A computes and sends to B (the *response*) y , either $y = r$ (if $e = 0$) or $y = rs \bmod n$ (if $e = 1$).
- (d) B rejects the proof if $y = 0$, and otherwise accepts upon verifying $y^2 \equiv x \cdot v^e \pmod{n}$. (Depending on e , $y^2 = x$ or $y^2 = xv \bmod n$, since $v = s^2 \bmod n$. Note that checking for $y = 0$ precludes the case $r = 0$.)

Protocol 10.24 may be explained and informally justified as follows. The challenge (or *exam*) e requires that A be capable of answering two questions, one of which demonstrates her knowledge of the secret s , and the other an easy question (for honest provers) to prevent cheating. An adversary impersonating A might try to cheat by selecting any r and setting $x = r^2/v$, then answering the challenge $e = 1$ with a “correct” answer $y = r$; but would be unable to answer the exam $e = 0$ which requires knowing a square root of $x \bmod n$. A prover A knowing s can answer both questions, but otherwise can at best answer one of the two questions, and so has probability only $1/2$ of escaping detection. To decrease the probability of cheating arbitrarily to an acceptably small value of 2^{-t} (e.g., $t = 20$ or $t = 40$), the protocol is iterated t times, with B accepting A ’s identity only if all t questions (over t rounds) are successfully answered.

10.25 Note (*secret information revealed by A*) The response $y = r$ is independent of A ’s secret s , while the response $y = rs \bmod n$ also provides no information about s because the random r is unknown to B . Information pairs (x, y) extracted from A could equally well be simulated by a verifier B alone by choosing y randomly, then defining $x = y^2$ or $y^2/v \bmod n$. While this is not the method by which A would construct such pairs, such pairs (x, y) have a probability distribution which is indistinguishable from those A would produce; this establishes the zero-knowledge property. Despite the ability to simulate proofs, B is unable to impersonate A because B cannot predict the real-time challenges.

As a minor technical point, however, the protocol does reveal a bit of information: the answer $y = rs$ provides supporting evidence that v is indeed a square modulo n , and the soundness of the protocol allows one to conclude, after t successful iterations, that this is indeed the case.

(iv) General structure of zero-knowledge protocols

Protocol 10.24 illustrates the general structure of a large class of three-move zero-knowledge protocols:

$$\begin{aligned} A &\rightarrow B : && \text{witness} \\ A &\leftarrow B : && \text{challenge} \\ A &\rightarrow B : && \text{response} \end{aligned}$$

The prover claiming to be A selects a random element from a pre-defined set as its secret commitment (providing hidden randomization or “private coin tosses”), and from this computes an associated (public) witness. This provides initial randomness for variation from other protocol runs, and essentially defines a set of questions all of which the prover claims to be able to answer, thereby a priori constraining her forthcoming response. By protocol design, only the legitimate party A , with knowledge of A ’s secret, is truly capable of answering all the questions, and the answer to any one of these provides no information about

Handbook of Applied Cryptography by A. Menezes, P. van Oorschot and S. Vanstone.

A 's long-term secret. B 's subsequent challenge selects one of these questions. A provides its response, which B checks for correctness. The protocol is iterated, if necessary, to improve the bound limiting the probability of successful cheating.

Zero-knowledge interactive protocols thus combine the ideas of *cut-and-choose* protocols (this terminology results from the standard method by which two children share a piece of cake: one cuts, the other chooses) and challenge-response protocols. A responds to at most one challenge (question) for a given witness, and should not reuse any witness; in many protocols, security (possibly of long-term keying material) may be compromised if either of these conditions is violated.

10.4.2 Feige-Fiat-Shamir identification protocol

The basic version of the Fiat-Shamir protocol is presented as Protocol 10.24. This can be generalized, and the Feige-Fiat-Shamir (FFS) identification protocol (Protocol 10.26) is a minor variation of such a generalization. The FFS protocol involves an entity identifying itself by proving knowledge of a secret using a zero-knowledge proof; the protocol reveals no partial information whatsoever regarding the secret identification value(s) of A (cf. Definition 10.20). It requires limited computation (a small fraction of that required by RSA – see §10.4.5), and is thus well-suited for applications with low-power processors (e.g., 8-bit chipcard microprocessors).

10.26 Protocol Feige-Fiat-Shamir identification protocol

SUMMARY: A proves its identity to B in t executions of a 3-pass protocol.

1. *Selection of system parameters.* A trusted center T publishes the common modulus $n = pq$ for all users, after selecting two secret primes p and q each congruent to 3 mod 4, and such that n is computationally infeasible to factor. (Consequently, n is a Blum integer per §2.4.6, and -1 is a quadratic non-residue mod n with Jacobi symbol $+1$.) Integers k and t are defined as security parameters (see Note 10.28).
2. *Selection of per-entity secrets.* Each entity A does the following.
 - (a) Select k random integers $s_1, s_2, \dots, s_k$ in the range $1 \leq s_i \leq n-1$, and k random bits $b_1, \dots, b_k$. (For technical reasons, $\gcd(s_i, n) = 1$ is required, but is almost surely guaranteed as its failure allows factorization of n .)
 - (b) Compute $v_i = (-1)^{b_i} \cdot (s_i^2)^{-1} \bmod n$ for $1 \leq i \leq k$. (This allows v_i to range over all integers coprime to n with Jacobi symbol $+1$, a technical condition required to prove that no secret information is “leaked”; by choice of n , precisely one signed choice for v_i has a square root.)
 - (c) A identifies itself by non-cryptographic means (e.g., photo id) to T , which thereafter registers A 's public key $(v_1, \dots, v_k; n)$, while only A knows its private key $(s_1, \dots, s_k)$ and n . (To guarantee the bounded probability of attack specified per Note 10.28, T may confirm that each v_i indeed does have Jacobi symbol $+1$ relative to n .) This completes the one-time set-up phase.
3. *Protocol messages.* Each of t rounds has three messages with form as follows.

$$A \rightarrow B : x (= \pm r^2 \bmod n) \quad (1)$$

$$A \leftarrow B : (e_1, \dots, e_k), e_i \in \{0, 1\} \quad (2)$$

$$A \rightarrow B : y (= r \cdot \prod_{e_j=1} s_j \bmod n) \quad (3)$$
4. *Protocol actions.* The following steps are executed t times; B accepts A 's identity if all t rounds succeed. Assume B has A 's authentic public key $(v_1, \dots, v_k; n)$; otherwise, a certificate may be sent in message (1), and used as in Protocol 10.36.

- (a) A chooses a random integer r , $1 \leq r \leq n - 1$, and a random bit b ; computes $x = (-1)^b \cdot r^2 \bmod n$; and sends x (the *witness*) to B .
- (b) B sends to A (the *challenge*), a random k -bit vector $(e_1, \dots, e_k)$.
- (c) A computes and sends to B (the *response*): $y = r \cdot \prod_{j=1}^k s_j^{e_j} \bmod n$ (the product of r and those s_j specified by the challenge).
- (d) B computes $z = y^2 \cdot \prod_{j=1}^k v_j^{e_j} \bmod n$, and verifies that $z = \pm x$ and $z \neq 0$. (The latter precludes an adversary succeeding by choosing $r = 0$.)

10.27 Example (*Feige-Fiat-Shamir protocol with artificially small parameters*)

1. The trusted center T selects the primes $p = 683$, $q = 811$, and publishes $n = pq = 553913$. Integers $k = 3$ and $t = 1$ are defined as security parameters.
2. Entity A does the following.
 - (a) Selects 3 random integers $s_1 = 157$, $s_2 = 43215$, $s_3 = 4646$, and 3 bits $b_1 = 1$, $b_2 = 0$, $b_3 = 1$.
 - (b) Computes $v_1 = 441845$, $v_2 = 338402$, and $v_3 = 124423$.
 - (c) A 's public key is $(441845, 338402, 124423; 553913)$ and private key is $(157, 43215, 4646)$.
3. See Protocol 10.26 for a summary of the messages exchanged.
4.
 - (a) A chooses $r = 1279$, $b = 1$, computes $x = 25898$, and sends this to B .
 - (b) B sends to A the 3-bit vector $(0, 0, 1)$.
 - (c) A computes and sends to B $y = r \cdot s_3 \bmod n = 403104$.
 - (d) B computes $z = y^2 \cdot v_3 \bmod n = 25898$ and accepts A 's identity since $z = +x$ and $z \neq 0$. $\square$

10.28 Note (*security of Feige-Fiat-Shamir identification protocol*)

- (i) *probability of forgery*. Protocol 10.26 is provably secure against chosen message attack in the following sense: provided that factoring n is difficult, the best attack has a probability 2^{-kt} of successful impersonation.
- (ii) *security assumption required*. The security relies on the difficulty of extracting square roots modulo large composite integers n of unknown factorization. This is equivalent to that of factoring n (see Fact 3.46).
- (iii) *zero-knowledge and soundness*. The protocol is, relative to a trusted server, a (sound) zero-knowledge proof of knowledge provided $k = O(\log \log n)$ and $t = \Theta(\log n)$. See Remark 10.34 regarding the practical significance of such constraints. A simplistic view for fixed k is that the verifier, interested in soundness, favors larger t (more iterations) for a decreased probability of fraud; while the prover, interested in zero-knowledge, favors smaller t .
- (iv) *parameter selection*. Choosing k and t such that $kt = 20$ allows a 1 in a million chance of impersonation, which suffices in the case that an identification attempt requires a personal appearance by a would-be impersonator (see §10.5). Computation, memory, and communication can be traded off; $1 \leq k \leq 18$ was originally suggested as appropriate. Specific parameter choices might be, for security 2^{-20} : $k = 5, t = 4$; for 2^{-30} : $k = 6, t = 5$.
- (v) *security trade-off*. Both computation and communication may be reduced by trading off security parameters to yield a single iteration ($t = 1$), holding the product kt constant and increasing k while decreasing t ; however, in this case the protocol is no longer a zero-knowledge proof of knowledge.

Handbook of Applied Cryptography by A. Menezes, P. van Oorschot and S. Vanstone.

10.29 Note (*modifications to Feige-Fiat-Shamir*)

- (i) As an alternative to step 1 of Protocol 10.26, each user may pick its own such modulus n . T is still needed to associate each user with its modulus.
- (ii) The communication complexity can be reduced if A sends B (e.g., 128 bits of) a hash value $h(x)$ instead of x in message (1), with B 's verification modified accordingly.
- (iii) The scheme can be made *identity-based* as follows (cf. §13.4.3). T assigns a distinguished identifying string I_A to each party A (e.g., A 's name, address, or other information which a verifier may wish to corroborate). A 's public values v_i , $1 \leq i \leq k$ are then derived by both T and other parties B as $v_i = f(I_A, i)$ using an appropriate function f . Then the trusted center, knowing the factorization of n , computes a square root s_i of each v_i and gives these to A .
As an example of f , consider, for a randomly chosen but known value c , $f(I_A, i) = I_A + i + c \bmod n$. Since a square root of $f_i = f(I_A, i)$ is required, any f_i with Jacobi symbol $-1 \bmod n$ may be multiplied by a fixed number with Jacobi symbol -1 . A non-residue f_i with Jacobi $+1$ may be either discarded (A must then indicate to B , e.g., in message (3), which values i allow computation of the v_j); or mapped to a residue via multiplication by -1 , again with an indication to B of this to allow computation of v_j . Note that both cases for dealing with a non-residue f_i with Jacobi $+1$ reveal some (non-useful) information.
- (iv) The *parallel version* of the protocol, in which each of three messages contains the respective data for all t rounds simultaneously, can be shown to be secure (it releases no "transferable information"), but for technical reasons loses the zero-knowledge property. Such parallel execution (as opposed to *sequential iteration*) in interactive proofs allows the probability of error (forgery) to be decreased without increasing the number of rounds.

10.30 Note (*converting identification to signature scheme*) The following general technique may be used to convert an identification scheme involving a witness-challenge-response sequence to a signature scheme: replace the random challenge e of the verifier by the one-way hash $e = h(x||m)$, of the concatenation of the witness x and the message m to be signed (h essentially plays the role of verifier). As this converts an interactive identification scheme to a non-interactive signature scheme, the bitsize of the challenge e must typically be increased to preclude off-line attacks on the hash function.

10.4.3 GQ identification protocol

The Guillou-Quisquater (GQ) identification scheme (Protocol 10.31) is an extension of the Fiat-Shamir protocol. It allows a reduction in both the number of messages exchanged and memory requirements for user secrets and, like Fiat-Shamir, is suitable for applications in which the claimant has limited power and memory. It involves three messages between a claimant A whose identity is to be corroborated, and a verifier B .

10.31 Protocol GQ identification protocol

SUMMARY: A proves its identity (via knowledge of s_A) to B in a 3-pass protocol.

1. *Selection of system parameters.*

- (a) An authority T , trusted by all parties with respect to binding identities to public keys, selects secret random RSA-like primes p and q yielding a modulus $n = pq$. (As for RSA, it must be computationally infeasible to factor n .)

©1997 by CRC Press, Inc. — See accompanying notice at front of chapter.

- (b) T defines a public exponent $v \geq 3$ with $\gcd(v, \phi) = 1$ where $\phi = (p-1)(q-1)$, and computes its private exponent $s = v^{-1} \bmod \phi$. (See Note 10.33.)
 - (c) System parameters (v, n) are made available (with guaranteed authenticity) for all users.
2. *Selection of per-user parameters.*
- (a) Each entity A is given a unique identity I_A , from which (the *redundant identity*) $J_A = f(I_A)$, satisfying $1 < J_A < n$, is derived using a known redundancy function f . (See Note 10.35. Assuming that factoring n is difficult implies $\gcd(J_A, \phi) = 1$.)
 - (b) T gives to A the secret (*accreditation data*) $s_A = (J_A)^{-s} \bmod n$.
3. *Protocol messages.* Each of t rounds has three messages as follows (often $t = 1$).
- $$\begin{aligned} A \rightarrow B: & \quad I_A, x = r^v \bmod n & (1) \\ A \leftarrow B: & \quad e \text{ (where } 1 \leq e \leq v) & (2) \\ A \rightarrow B: & \quad y = r \cdot s_A^e \bmod n & (3) \end{aligned}$$
4. *Protocol actions.* A proves its identity to B by t executions of the following; B accepts the identity only if all t executions are successful.
- (a) A selects a random secret integer r (the *commitment*), $1 \leq r \leq n-1$, and computes (the *witness*) $x = r^v \bmod n$.
 - (b) A sends to B the pair of integers (I_A, x) .
 - (c) B selects and sends to A a random integer e (the *challenge*), $1 \leq e \leq v$.
 - (d) A computes and sends to B (the *response*) $y = r \cdot s_A^e \bmod n$.
 - (e) B receives y , constructs J_A from I_A using f (see above), computes $z = J_A^e \cdot y^v \bmod n$, and accepts A 's proof of identity if both $z = x$ and $z \neq 0$. (The latter precludes an adversary succeeding by choosing $r = 0$.)

10.32 Example (*GQ identification protocol with artificially small parameters and $t = 1$*)

1. (a) The authority T selects primes $p = 569$, $q = 739$, and computes $n = pq = 420491$.
- (b) T computes $\phi = (p-1)(q-1) = 419184$, selects $v = 54955$, and computes $s = v^{-1} \bmod \phi = 233875$.
- (c) System parameters $(54955, 420491)$ are made available for all users.
2. (a) Suppose that A 's redundant identity is $J_A = 34579$.
- (b) T gives to A the accreditation data $s_A = (J_A)^{-s} \bmod n = 403154$.
3. See Protocol 10.31 for a summary of the messages exchanged.
4. (a) A selects $r = 65446$ and computes $x = r^v \bmod n = 89525$.
- (b) A sends to B the pair $(I_A, 89525)$.
- (c) B sends to A the random challenge $e = 38980$.
- (d) A sends $y = r \cdot s_A^e \bmod n = 83551$ to B .
- (e) B computes $z = J_A^e \cdot y^v \bmod n = 89525$ and accepts A 's identity since $z = x$ and $z \neq 0$. $\square$

10.33 Note (*security of GQ identification protocol*)

- (i) *probability of forgery.* In Protocol 10.31, v determines the security level (cf. Fiat-Shamir where $v = 2$ but there are many rounds); some values such as $v = 2^{16} + 1$ may offer computational advantages. A fraudulent claimant can defeat the protocol with a 1 in v chance by guessing e correctly a priori (and then forming $x = J_A^e \cdot y^v$ as the verifier would). The recommended bitlength of v thus depends on the environment under which attacks could be mounted (see §10.5).

- (ii) *security assumption required.* Extracting v^{th} roots modulo the composite integer n (i.e., solving the RSA problem – §3.3) appears necessary to defeat the protocol; this is no harder than factoring n (Fact 3.30), and appears computationally intractable without knowing the factors of n .
- (iii) *soundness.* In practice, GQ with $t = 1$ and a k -bit prime v is often suggested. For generalized parameters (n, v, t) , the probability of forgery is v^{-t} . If v is constant, then technically for soundness, t must grow asymptotically faster than $\log \log n$. (For soundness, $v^{-t} = O(e^{-kt})$ must be smaller than inverse-polynomial in $\log n$; only polynomial security is provided if for a constant c , $v^t = O((\log n)^c)$. See also Remark 10.34.)
- (iv) *zero-knowledge property.* In opposition to the soundness requirement, for GQ to be zero-knowledge apparently requires $tv = O((\log n)^c)$ for constant c , imposing an upper bound on t asymptotically: for v constant, t must be no larger than polynomial in $\log n$.

10.34 Remark (*asymptotic concepts vs. practical protocols*) The asymptotic conditions for soundness specified in Note 10.33 have little meaning in practice, e.g., because big-O notation is not applicable once fixed values are assigned to parameters. Indeed, zero-knowledge is a theoretical concept; while complexity-theoretic definitions offer guidance in selecting practical security parameters, their significance diminishes when parameters are fixed. Regarding Note 10.33, if $t = 1$ is viewed as the instantiation of a non-constant parameter (e.g., the iterated logarithm of n), then $t = 1$ will suffice for all practical purposes; consider $n = 1024$, $t = \lceil \lg^4 n \rceil = 1$.

10.35 Note (*redundancy function for identity-based GQ*)

- (i) The protocol as given is an identity-based version (cf. Note 10.29), where A 's public key is reconstructed from identifier I_A sent in message (1). Alternatively, a certified public key may be used, distributed in a certificate as per Protocol 10.36.
- (ii) One example of the redundancy function f is the redundancy mapping of the preprocessing stage of ISO/IEC 9796 (see §11.3.5). A second example is a single function value of f as in Note 10.29, for an appropriate value i .
- (iii) The purpose of the redundancy is to preclude an adversary computing false accreditation data corresponding to a plausible identity; this would be equivalent to forging a certificate in certificate-based schemes.

10.4.4 Schnorr identification protocol

The Schnorr identification protocol is an alternative to the Fiat-Shamir and GQ protocols. Its security is based on the intractability of the discrete logarithm problem. The design allows pre-computation, reducing the real-time computation for the claimant to one multiplication modulo a prime q ; it is thus particularly suitable for claimants of limited computational ability. A further important computational efficiency results from the use of a subgroup of order q of the multiplicative group of integers modulo p , where $q|(p-1)$; this also reduces the required number of transmitted bits. Finally, the protocol was designed to require only three passes, and a low communications bandwidth (e.g., compared to Fiat-Shamir).

The basic idea is that A proves knowledge of a secret a (without revealing it) in a time-variant manner (depending on a challenge e), identifying A through the association of a with the public key v via A 's authenticated certificate.

©1997 by CRC Press, Inc. — See accompanying notice at front of chapter.

10.36 Protocol Schnorr identification protocol

SUMMARY: A proves its identity to B in a 3-pass protocol.

1. *Selection of system parameters.*
 - (a) A suitable prime p is selected such that $p - 1$ is divisible by another prime q . (Discrete logarithms modulo p must be computationally infeasible – see §3.6; e.g., $p \approx 2^{1024}$, $q \geq 2^{160}$.)
 - (b) An element β is chosen, $1 \leq \beta \leq p - 1$, having multiplicative order q . (For example, for α a generator mod p , $\beta = \alpha^{(p-1)/q} \bmod p$; see Note 4.81.)
 - (c) Each party obtains an authentic copy of the system parameters (p, q, β) and the verification function (public key) of the trusted party T , allowing verification of T 's signatures $S_T(m)$ on messages m . (S_T involves a suitable known hash function prior to signing, and may be any signature mechanism.)
 - (d) A parameter t (e.g., $t \geq 40$), $2^t < q$, is chosen (defining a security level 2^t).
2. *Selection of per-user parameters.*
 - (a) Each claimant A is given a unique identity I_A .
 - (b) A chooses a private key a , $0 \leq a \leq q - 1$, and computes $v = \beta^{-a} \bmod p$.
 - (c) A identifies itself by conventional means (e.g., passport) to T , transfers v to T with integrity, and obtains a certificate $\text{cert}_A = (I_A, v, S_T(I_A, v))$ from T binding I_A with v .
3. *Protocol messages.* The protocol involves three messages.

$$\begin{aligned} A \rightarrow B : & \text{cert}_A, x = \beta^r \bmod p & (1) \\ A \leftarrow B : & e \text{ (where } 1 \leq e \leq 2^t < q) & (2) \\ A \rightarrow B : & y = ae + r \bmod q & (3) \end{aligned}$$
4. *Protocol actions.* A identifies itself to verifier B as follows.
 - (a) A chooses a random r (the *commitment*), $1 \leq r \leq q - 1$, computes (the *witness*) $x = \beta^r \bmod p$, and sends (1) to B .
 - (b) B authenticates A 's public key v by verifying T 's signature on cert_A , then sends to A a (never previously used) random e (the *challenge*), $1 \leq e \leq 2^t$.
 - (c) A checks $1 \leq e \leq 2^t$ and sends B (the *response*) $y = ae + r \bmod q$.
 - (d) B computes $z = \beta^y v^e \bmod p$, and accepts A 's identity provided $z = x$.

10.37 Example (*Schnorr identification protocol with artificially small parameters*)

1.
 - (a) The prime $p = 48731$ is selected, where $p - 1$ is divisible by the prime $q = 443$.
 - (b) A generator mod 48731 is $\alpha = 6$; β is computed as $\alpha^{(p-1)/q} \bmod p = 11444$.
 - (c) The system parameters are (48731, 443, 11444).
 - (d) The parameter $t = 8$ is chosen.
2. (b) A chooses a private key $a = 357$ and computes $v = \beta^{-a} \bmod p = 7355$.
3. See Protocol 10.36 for a summary of the messages exchanged.
4.
 - (a) A chooses $r = 274$ and sends $x = \beta^r \bmod p = 37123$ to B .
 - (b) B sends to A the random challenge $e = 129$.
 - (c) A sends B the number $y = ae + r \bmod q = 255$.
 - (d) B computes $z = \beta^y v^e \bmod p = 37123$ and accepts A 's identity since $z = x$. □

10.38 Note (*security of Schnorr identification protocol*)

- (i) *probability of forgery.* In Protocol 10.36, t must be sufficiently large to make the probability 2^{-t} of correctly guessing the challenge e negligible. $t = 40$, $q \geq 2^{2t} = 2^{80}$ was originally suggested in the case that a response is required within seconds (see §10.5); larger q may be necessary to preclude time-memory trade-offs, and $q \geq 2^{160}$ is recommended to preclude other off-line discrete log attacks. Correctly guessing e allows an adversary to impersonate A by choosing any y , sending $x = \beta^y v^e \bmod p$ to B in (1), then sending y in (3).
- (ii) *soundness.* It can be shown that the protocol is a proof of knowledge of a , i.e., any party completing the protocol as A must be capable of computing a . Informally, the protocol reveals “no useful information” about a because x is a random number, and y is perturbed by the random number r . (However, this does not prove that adversarial discovery of a is difficult.)
- (iii) *zero-knowledge property.* The protocol is not zero-knowledge for large e , because through interaction, B obtains the solution (x, y, e) to the equation $x = \beta^y v^e \bmod p$, which B itself might not be able to compute (e.g., if e were chosen to depend on x).

10.39 Note (*reducing transmission bandwidth*) The number of bits transmitted in the protocol can be reduced by replacing x in message (1) by t pre-specified bits of x (e.g., the least significant t bits), and having B compare this to t corresponding bits of z .

10.4.5 Comparison: Fiat-Shamir, GQ, and Schnorr

The protocols of Feige-Fiat-Shamir, Guillou-Quisquater, and Schnorr all provide solutions to the identification problem. Each has relative advantages and disadvantages with respect to various performance criteria and for specific applications. To compare the protocols, a typical set of selected parameters must be chosen for each providing comparable estimated security levels. The protocols may then be compared based on the following criteria:

1. *communications:* number of messages exchanged, and total bits transferred;
2. *computations:* number of modular multiplications for each of prover and verifier (noting on-line and off-line computations);
3. *memory:* storage requirements for secret keys (and signature size, in the case of signature schemes);
4. *security guarantees:* comparisons should consider security against forgery by guessing (soundness), possible disclosure of secret information (zero-knowledge property), and status regarding provable security; and
5. *trust required in third party:* variations of the protocols may require different trust assumptions in the trusted party involved.

The number of criteria and potential parameter choices precludes a comparison which is both definitive and concise. The following general comments may, however, be made.

1. *computational efficiency.* Fiat-Shamir requires between one and two orders of magnitude fewer full modular multiplications (steps) by the prover than an RSA private-key operation (cf. §10.3.3). When $kt = 20$ and n is 512 bits, Fiat-Shamir uses from about 11 to about 30 steps ($k = 20$, $t = 1$; and $k = 1$, $t = 20$); GQ requires about 60 steps (for $t = 1$, $m = 20 = \log_2(v)$), or somewhat fewer if v has low Hamming weight; and full exponentiation in unoptimized RSA takes 768 steps.

2. *off-line computations.* Schnorr identification has the advantage of requiring only a single on-line modular multiplication by the claimant, provided exponentiation may be done as a precomputation. (Such a trade-off of on-line for off-line computation is possible in some applications; in others, the total computation must be considered.) However, significant computation is required by the verifier compared to Fiat-Shamir and GQ.
3. *bandwidth and memory for secrets.* GQ allows the simultaneous reduction of both memory (parameter k) and transmission bandwidth (parameter t) with $k = t = 1$, by introducing the public exponent $v > 2$ with the intention that the probability of successful cheating becomes v^{-kt} ; this simultaneous reduction is not possible in Fiat-Shamir, which requires k user secrets and t iterations for an estimated security (probability of cheating) of 2^{-kt} . Regarding other tradeoffs, see Note 10.28.
4. *security assumptions.* The protocols require the assumptions that the following underlying problems are intractable, for a composite (RSA) integer n : Fiat-Shamir – extracting square roots mod n ; GQ – extracting v^{th} roots mod n (i.e., the RSA problem); Schnorr identification – computing discrete logs modulo a prime p .

10.5 Attacks on identification protocols

The methods an adversary may employ in an attempt to defeat identification protocols are a subset of those discussed in Chapter 12 for authenticated key establishment, and the types of adversaries may be similarly classified (e.g., passive vs. active, insider vs. outsider); for a discussion of attacks on simple password schemes, see §10.2.2. Identification is, however, less complex than authenticated key establishment, as there is no issue of an adversary learning a previous session key, or forcing an old key to be reused. For conciseness, the following definitions are made:

1. *impersonation:* a deception whereby one entity purports to be another.
2. *replay attack:* an impersonation or other deception involving use of information from a single previous protocol execution, on the same or a different verifier. For stored files, the analogue of a replay attack is a *restore* attack, whereby a file is replaced by an earlier version.
3. *interleaving attack:* an impersonation or other deception involving selective combination of information from one or more previous or simultaneously ongoing protocol executions (*parallel sessions*), including possible origination of one or more protocol executions by an adversary itself.
4. *reflection attack:* an interleaving attack involving sending information from an ongoing protocol execution back to the originator of such information.
5. *forced delay:* a forced delay occurs when an adversary intercepts a message (typically containing a sequence number), and relays it at some later point in time. Note the delayed message is not a replay.
6. *chosen-text attack:* an attack on a challenge-response protocol wherein an adversary strategically chooses challenges in an attempt to extract information about the claimant's long-term key.

Chosen-text attacks are sometimes referred to as using the claimant as an *oracle*, i.e., to obtain information not computable from knowledge of a claimant's public key alone. The attack may involve chosen-plaintext if the claimant is required to sign,

Handbook of Applied Cryptography by A. Menezes, P. van Oorschot and S. Vanstone.

encrypt, or MAC the challenge, or chosen-ciphertext if the requirement is to decrypt a challenge.

Potential threats to identification protocols include impersonation by any of the following attacks: replay, interleaving, reflection, or forced delay. Impersonation is also trivial if an adversary is able to discover an entity's long-term (secret or private) keying material, for example, using a chosen-text attack. This may be possible in protocols which are not zero-knowledge, because the claimant uses its private key to compute its response, and thus a response may reveal partial information. In the case of an active adversary, attacks may involve the adversary itself initiating one or more new protocol runs, and creating, injecting, or otherwise altering new or previous messages. Table 10.3 summarizes counter-measures for these attacks.

Type of attack	Principles to avoid attack
replay	use of challenge-response techniques; use of nonces; embed target identity in response
interleaving	linking together all messages from a protocol run (e.g., using chained nonces)
reflection	embed identifier of target party in challenge responses; construct protocols with each message of different form (avoid message symmetries); use of uni-directional keys
chosen-text	use of zero-knowledge techniques; embed in each challenge response a self-chosen random number (<i>confounder</i>)
forced delay	combined use of random numbers with short response time-outs; timestamps plus appropriate additional techniques

Table 10.3: Identification protocol attacks and counter-measures.

10.40 Remark (*use of keys for multiple purposes*) Caution is advised if any cryptographic key is used for more than one purpose. For example, using an RSA key for both entity authentication and signatures may compromise security by allowing a chosen-text attack. Suppose authentication here consists of B challenging A with a random number r_B RSA-encrypted under A 's public key, and A is required to respond with the decrypted random number. If B challenges A with $r_B = h(x)$, A 's response to this authentication request may (unwittingly) provide to B its RSA signature on the hash value of the (unknown to A) message x . See also Example 9.88, where a DES key used for both CBC encryption and CBC-MAC leads to a security flaw; and Remark 13.32.

10.41 Remark (*adversary acting "as a wire"*) In any identification protocol between A and B , an adversary C may step into the communications path and simply relay (without changing) the messages between legitimate parties A and B , itself acting as a part of the communications link. Typically in practice, this is not considered a true "attack", in the sense that it does not alter the aliveness assurance delivered by the protocol; however, in some special applications, this may be a concern (see Remark 10.42).

10.42 Remark (*grandmaster postal-chess problem*) Identification protocols do not provide assurances about the physical location of the authenticated party. Therefore, Remark 10.41 notwithstanding, a concern may arise in the special case that the following is possible: an adversary C attempts to impersonate B , is challenged (to prove it is B) by A , and is able to

relay (in real time, without detection or noticeable delay, and pretending to be *A*) the challenge on to the real *B*, get a proper response from *B*, and pass this response along back to *A*. In this case, additional measures are necessary to prevent a challenged entity from eliciting aid in computing responses. This is related to the so-called *grandmaster postal-chess problem*, whereby an amateur's chess rating may unfairly be improved by engaging in two simultaneous chess games with distinct grandmasters, playing black in one game and white in the second, and using the grandmaster's moves from each game in the other. Either two draws, or a win and a loss, are guaranteed, both of which will improve the amateur's rating.

For further discussion of protocol attacks including specific examples of flawed entity authentication protocols, see §12.9.

(i) Maintaining authenticity

Identification protocols provide assurances corroborating the identity of an entity only at a given instant in time. If the continuity of such an assurance is required, additional techniques are necessary to counteract active adversaries. For example, if identification is carried out at the beginning of a communications session to grant communications permissions, a potential threat is an adversary who "cuts in" on the communications line immediately after the successful identification of the legitimate party. Approaches to prevent this include:

1. performing re-authentication periodically, or for each discrete resource requested (e.g., each file access). A remaining threat here is an adversary who "steps out" every time re-authentication is performed, allowing the legitimate party to perform this task, before re-entering.
2. tying the identification process to an ongoing integrity service. In this case, the identification process should be integrated with a key establishment mechanism, such that a by-product of successful identification is a session key appropriate for use in a subsequent ongoing integrity mechanism.

(ii) Security level required for on-line vs. off-line attacks

The security level required for identification protocols depends on the environment and the specific application at hand. The probability of success of "guessing attacks" should be considered, and distinguished from the amount of computation required to mount on-line or off-line attacks (using the best techniques known). Some illustrative notes follow (see also Note 10.28).

1. *Local attacks*. Selecting security parameters which limit the probability of successful impersonation of a guessing attack (an adversary simply guesses a legitimate party's secret) to a 1 in 2^{20} chance (20 bits of security) may suffice if, for each attempted impersonation, a local appearance is required by the would-be impersonator and there is a penalty for failed attempts. Depending on the potential loss resulting relative to the penalty, 10 to 30 bits or more of security may be required.
2. *Remote attacks*. A higher level of security is required in environments where unlimited identification attempts, each involving minimal computational effort, are possible by remote electronic communications, by an anonymous claimant interacting with an on-line system, with no penalties for failed attempts. 20 to 40 bits of security or more may be called for here, unless the number of interactions may be somehow limited.
3. *Off-line or non-interactive attacks*. Selecting security parameters such that an attack requires 2^{40} computations in real-time (during a protocol execution) may be acceptable, but a bound of 2^{60} to 2^{80} computations (the latter should be adequate in all

Handbook of Applied Cryptography by A. Menezes, P. van Oorschot and S. Vanstone.

cases) may be called for if the computations can be carried out off-line, and the attack is *verifiable* (i.e., the adversary can confirm, before interacting with the on-line system, that his probability of successful impersonation is near 1; or can recover a long-term secret by off-line computations subsequent to an interaction).

10.6 Notes and further references

§10.1

Davies and Price [308] and Ford [414] provide extensive discussion of authentication and identification; see also the former for biometric techniques, as well as Everett [380]. The comprehensive survey on login protocols by de Waleffe and Quisquater [319] is highly recommended. Crépeau and Goutier provide a lucid concise summary of user identification techniques with Brassard [192]. For standardized entity authentication mechanisms, see ISO/IEC 9798 [598, 599, 600, 601, 602].

§10.2

See the §9.2 notes on page 377 for historical discussion of using a one-way function (*one-way cipher*) for “encrypted” password files. Morris and Thompson [907] introduce the notion of password salting in their 1979 report on UNIX passwords; in one study of 3289 user passwords unconstrained by password rules, 86% fell within an easily-searched subset of passwords. Feldmeier and Karn [391] give an update 10 years later, indicating 30% of passwords they encountered fell to their attack using a precomputed encrypted dictionary, sorted on tapes by salt values. See also Klein [680] and Lomas et al. [771]. Password salting is related to randomized encryption; the idea of padding plaintext with random bits before encryption may also be used to prevent *forward search* attacks on public-key encryption with small plaintext spaces. Password rules and procedures have been published by the U.S. Departments of Commerce [399] and Defense [334].

Methods for computing password-derived keys (§10.2.4) are specified in the Kerberos Authentication Service [1041] and PKCS #5 [1072]. A concern related to password-derived keys is that known plaintext allows password-guessing attacks; protocols specifically designed to prevent such attacks are mentioned in Chapter 12 notes on §12.6. The idea of chaining one-time passwords by a one-way function (Protocol 10.6) is due to Lamport [739]; for related practical applications, see RFC 1938 [1047]. Davies and Price [308, p.176] note a questionnaire-based identification technique related to fixed challenge-response tables, wherein the user is challenged by a random subset of previously answered questions.

§10.3

Needham and Schroeder [923] stimulated much early work in the area of authentication protocols in the late 1970s, and Needham was again involved with Burrows and Abadi [227] in the BAN logic work which stimulated considerable interest in protocol analysis beginning in the late 1980s; see Chapter 12 notes for further discussion.

Gong [501] provides an overview of both time variant parameters and message replay; see also Neuman and Stubblebine [925], and the annexes of parts of ISO/IEC 9798 (e.g., [600]). For security arguments against the use of timestamps and a discussion of implementation difficulties, see Bellovin and Merritt [103]; Gaarder and Snekenes [433]; Diffie, van Oorschot, and Wiener [348]; and Gong [500], who considers postdated timestamps. See also §12.3 notes. Lam and Beth [734] note that timestamp-based protocols are appropriate

for connectionless interactions whereas challenge-response suits connection-oriented communications, and suggest challenge-response techniques be used to securely synchronize timeclocks with applications themselves using timestamp-based authentication.

ISO/IEC 9798 [598] parts 2 through 5 specify entity authentication protocols respectively based on symmetric encryption [599], digital signatures [600], keyed one-way functions [601], and zero-knowledge techniques [602]; a subset of these are presented in this chapter. FIPS 196 [407] is a subset of 9798-3 containing the unilateral and mutual authentication protocols involving challenge-response with random numbers.

Several parts of 9798 were influenced by the SKID2 and SKID3 (*Secret Key IDentification*) protocols from the RACE/RIPE project [178], which leave the keyed hash function unspecified but recommend RIPE-MAC with 64-bit random-number challenges. Diffie [342, 345] notes that two-pass challenge-response identification based on encryption and random challenges has been used since the 1950s in military *Identification Friend or Foe* (IFF) systems to distinguish friendly from hostile aircraft. Mao and Boyd [781] discuss the danger of improperly using encryption in authentication protocols, specifically the CBC mode without an integrity mechanism (cf. Remark 10.16). Stubblebine and Gligor [1179] discuss attacks involving this same mode; see also the much earlier paper by Akl [20].

Davies and Price [308] give a concise discussion of password generators. The identification technique in §10.3.3(i) based on public-key decryption and witness is derived from a Danish contribution to the 4th Working Draft of ISO/IEC 9798-5, specifying a protocol called COMSET and motivated in part by Brandt et al. [188], and related to ideas noted earlier by Blum et al. [163].

§10.4

A refreshingly non-mathematical introduction to zero-knowledge proofs is provided by Quisquater, Guillou, and Berson [1020], who document the secret of Ali Baba's legendary cave, and its rediscovery by Mick Ali. Mitropoulos and Meijer [883] give an exceptionally readable and comprehensive survey (circa 1990) of interactive proofs and zero knowledge, with a focus on identification. Other overviews include Johnson [641]; Stinson [1178, Ch.13]; and Brassard, Chaum, and Crépeau [193] (or [192]) for a discussion of *minimum disclosure* proofs, based on *bit commitment* and the primitive of a *blob*. Brassard and Crépeau [195] provide a user-friendly discussion of various definitions of zero-knowledge, while Goldreich and Oren [475] examine properties and relationships between various definitions of ZK proof systems.

Rabin [1022] employed the idea of *cut-and-choose protocols* for cryptographic applications as early as 1978. While Babai (with Moran) [60, 61] independently developed a theory of randomized interactive proofs known as *Arthur-Merlin games* in an attempt to “formalize the notion of efficient provability by overwhelming statistical evidence”, interactive proof systems and the notion of zero-knowledge (ZK) proofs were formalized in 1985 by Goldwasser, Micali, and Rackoff [481] in the context of an interactive *proof of membership* of a string x in a language $\mathcal{L}$; they showed that the languages of quadratic-residues and of quadratic non-residues each have ZK interactive proof (ZKIP) systems revealing only a single bit of knowledge, namely, that $x \in \mathcal{L}$. Goldreich, Micali, and Wigderson [473, 474] prove likewise for graph non-isomorphism (known not to be in NP) and graph isomorphism, and that assuming the existence of secure encryption schemes, every language in NP has a ZKIP; see also Chaum [244], and Brassard and Crépeau [194].

Motivated by cryptographic applications and identification in particular, Feige, Fiat, and Shamir [383] adapted the concepts of interactive proofs of membership to interactive *proofs*

Handbook of Applied Cryptography by A. Menezes, P. van Oorschot and S. Vanstone.

of knowledge, including reformulated definitions for completeness, soundness, and zero-knowledge; while proofs of membership reveal one bit of set membership information, proofs of knowledge reveal only one bit about the prover's *state* of knowledge. The definitions given in §10.4.1 are based on these. These authors refine the original scheme of Fiat and Shamir [395] to yield that of Protocol 10.26; both may be converted to identity-based schemes (Note 10.29) in the sense of Shamir [1115]. The Fiat-Shamir scheme is related to (but more efficient than) an earlier protocol for proving quadratic residuosity (presented at Eurocrypt'84, but unpublished) by Fischer, Micali, and Rackoff [412]. The Fiat-Shamir protocol as per Protocol 10.24 includes an improvement noted by Desmedt et al. [340] to avoid inverses in the derivation of user secrets; this optimization may also be made to Protocol 10.26.

Related to definitions in §10.4.1, Bellare and Goldreich [87] noted that Goldwasser, Micali, and Rackoff [481] did not formally propose a definition for a proof of knowledge, and suggested that the formal definitions of Feige, Fiat, and Shamir [383] and Tompa and Woll [1194] were unsatisfactory for some applications. To address these issues they proposed a new definition, having some common aspects with that of Feige and Shamir [384], but offering additional advantages.

Micali and Shamir [868] provide preliminary notes on reducing computation in the Fiat-Shamir protocol by choosing the public keys v_i , $1 \leq i \leq k$ to be the first k prime numbers; each user then has an independent modulus n . A modification of Fiat-Shamir identification by Ong and Schnorr [957] decreases computational complexity, signature size, and the number of communications required, condensing t Fiat-Shamir iterations into one iteration while leaving each user with k private keys (cf. the $k = 1$ extension below); for computational efficiency, they suggest using as secret keys (not too) small integers.

The idea of generalizing Fiat-Shamir identification in other ways, including "replacing square roots by cubic or higher roots", was suggested in the original paper; using higher roots allows users to reduce their number of private keys k , including to the limiting case $k = 1$. Guillou and Quisquater [524] proposed a specific formulation of this idea of "using deep coin tosses" as the GQ scheme (Protocol 10.31); apparently independently, Ohta and Okamoto [945, 944] proposed a similar formulation, including security analysis.

The Ohta-Okamoto (OO) version of this *extended Fiat-Shamir* scheme differs from the GQ version (Protocol 10.31) as follows: (1) in OO, rather than T computing s_A from identity I_A , A chooses its own secret $s_A \in \mathbb{Z}_n$ and publishes $I_A = s_A^v \bmod n$; and (2) the verification relation $x \equiv J_A^e \cdot y^v \pmod{n}$ becomes $y^v \equiv x \cdot I_A^e$. OO is more general in that, as originally proposed, it avoids the GQ (RSA) constraint that $\gcd(v, \phi(n)) = 1$. Subsequent analysis by Burmester and Desmedt [221] suggests that additional care may be required when v is not prime. While the OO version precludes an identity-based variation, a further subsequent version of extended Fiat-Shamir (GQ variation) by Okamoto [949] ("Scheme 3" of 5 protocols therein) is provably as secure as factoring, only slightly less efficient, and is amenable to an identity-based variation.

The zero-knowledge interactive protocols of Chaum et al. [248, 249] for proving possession of discrete logarithms, provided a basis for Protocol 10.36 which is due to Schnorr [1097, 1098]. Schnorr also proposed a preprocessing scheme to reduce real-time computation, but see de Rooij [314] regarding its security. The Schnorr identification and signature schemes must not both be used with the same parameters β, p [1098] (cf. Remark 10.40). Schnorr's protocol is related to the log-based identification scheme of Beth [123] also proven to be zero-knowledge. Burmester et al. [223] analyze (cf. Note 10.33) a generalized identification protocol encompassing all the well-known variations related to Fiat-Shamir and including

those of both Chaum et al. and Beth noted above. Van de Graaf and Peralta [1200] give a ZK interactive protocol for proving that a Blum integer is a Blum integer.

Brickell and McCurley [207] propose a modification of Schnorr's identification scheme, in which q is kept secret and exponent computations are reduced modulo $p - 1$ rather than q ; it has provable security if factoring $p - 1$ is difficult, and moreover security equivalent to that of Schnorr's scheme otherwise; a drawback is that almost 4 times as much computation is required by the claimant. Another variant of Schnorr's scheme by Girault [458, 461] was the first identity-based identification scheme based on discrete logs; it uses a composite modulus, and features the user choosing its own secret key, which remains unknown to the trusted party (cf. implicitly-certified public keys, §12.6.2). A further variation of Schnorr's identification protocol by Okamoto [949] ("Scheme 1") uses two elements β_1 and β_2 , of order q , and is provably secure, assuming the computational infeasibility of computing the $\mathbb{Z}_p$ discrete logarithm $\log_{\beta_1} \beta_2$ of β_2 relative to β_1 ; it does, however, involve some additional computation.

Aside from the above protocols based on the computational intractability of the standard number-theoretic problems (factoring and discrete logarithms), a number of very efficient identification protocols have more recently been proposed based on NP-hard problems. Shamir [1116] proposed a zero-knowledge identification protocol based on the NP-hard *permuted kernel problem*: given an $m \times n$ matrix A over $\mathbb{Z}_p$, p prime (and relatively small, e.g., $p = 251$), and an n -vector V , find a permutation π on $\{1, \dots, n\}$ such that $V_\pi \in \ker(A)$, where $\ker(A)$ is the kernel of A consisting of all n -vectors W such that $AW = [0 \dots 0] \bmod p$. Patarin and Chauvaud [966] discuss attacks on the permuted kernel problem which are feasible for the smallest of parameter choices originally suggested, while earlier less efficient attacks are presented by Baritaud et al. [73] and Georgiades [447]. Stern [1176] proposed a practical zero-knowledge identification scheme based on the NP-hard *syndrome decoding* problem, following an earlier less practical scheme of Stern [1174] based on intractable problems in coding theory. Stern [1175] proposed another practical identification scheme based on an NP-hard combinatorial *constrained linear equations* problem, offering a very short key length, which is of particular interest in specific applications. Pointcheval [983] proposed another such scheme based on the NP-hard *perceptrons problem*: given an $m \times n$ matrix M with entries ± 1 , find an n -vector y with entries ± 1 such that $My \geq 0$.

Goldreich and Krawczyk [469] pursue the fact that the original definition of ZK of Goldwasser, Micali, and Rackoff is not closed under sequential composition (this was noted earlier by D. Simon), establishing the importance of the stronger definitions of ZK formulated subsequently (e.g., *auxiliary-input* zero-knowledge – see Goldreich and Oren [475]), for which closure under sequential composition has been proven. They prove that even these strong formulations of ZK are not, however, closed under parallel composition (thus motivating the definition of weaker notions of zero-knowledge), and that 3-pass interactive ZK proofs of membership that are *black-box simulation* ZK exist only for languages in **BPP** (Definition 2.77); while the definition of "black-box simulation ZK" is more restrictive than the original definition of ZK, all known ZK protocols are ZK by this definition also. Consequently, protocols that are (formally) ZK are less practical than their corresponding 3-pass parallel versions.

As a replacement for the security requirement of zero knowledge in many protocols, Feige and Shamir [384] proposed *witness indistinguishability* and the related notion of *witness hiding protocols*. Unlike zero knowledge, witness indistinguishability is preserved under arbitrary composition of protocols.

Handbook of Applied Cryptography by A. Menezes, P. van Oorschot and S. Vanstone.

Methods have been proposed to reduce the communication complexity of essentially all customized identification protocols, including the use of hash values in the first message (cf. Note 10.29; Note 10.39). Girault and Stern [462] examine the security implications of the length of such hash values, note that collision-resistance of the hash function suffices for the typically claimed security levels, and examine further optimizations of the communication complexity of such protocols, including use of *r-collision resistant* hash functions.

Blum, Feldman, and Micali [163] introduced the idea of non-interactive (or more clearly: *mono-directional*) ZK proofs, separating the notions of interactive proof systems and zero-knowledge protocols; here the prover and verifier share a random string, and communication is restricted to one-way (or the prover may simply publish a proof, for verification at some future time). De Santis, Micali, and Persiano [317] improve these results employing a weaker complexity assumption; Blum et al. [162] provide a summary and further improvements. While the technique of Remark 10.30, due to Fiat and Shamir [395], allows a zero-knowledge identification scheme to be converted to a signature scheme, the latter cannot be a sound zero-knowledge signature scheme because the very simulatability of the identification which establishes the ZK property would allow signature forgery (e.g., see Okamoto [949]).

A further flavor of zero-knowledge (cf. Definition 10.22) is *statistical* (or *almost perfect*) zero-knowledge; here the probability distributions of the transcripts must be *statistically indistinguishable* (indistinguishable by an examiner with unlimited computing power but given only polynomially many samples). Pursuing other characterizations, interactive protocols in which the assurance a verifier obtains is based on some unproven assumption may be distinguished as *arguments* (see Brassard and Crépeau [195]), with *proofs* then required to be free of any unproven assumptions, although possibly probabilistic.

For performance comparisons and tradeoffs for the Fiat-Shamir, Guillou-Quisquater, and Schnorr schemes, see Fiat and Shamir [395], Schnorr [1098], Okamoto [949], and Lim and Lee [768], among others. For an overview of chipcard technology and the use thereof for identification, see Guillou, Ugon, and Quisquater [527]; an earlier paper on chipcards is by Guillou and Ugon [526]. Knobloch [681] describes a preliminary chipcard implementation of the Fiat-Shamir protocol.

§10.5

Bauspiess and Knobloch [78] discuss issues related to Remark 10.41, including taking over a communications line after entity authentication has completed. Bengio et al. [113] discuss implementation issues related to identification schemes such as the Fiat-Shamir protocol, including Remark 10.42. Classes of replay attacks are discussed in several papers, e.g., see Syverson [1182] and the ISO/IEC 10181-2 authentication framework [610]. For further references on the analysis of entity authentication protocols and attacks, see the §12.9 notes.