Cryptology

Prof. Ronald L. Rivest
MIT Laboratory for Computer Science
545 Technology Square
Cambridge, Mass. 02139

Cryptology

Abstract

Cryptology has advanced tremendously since 1976; this chapter provides a brief overview of the current state-of-the-art in the field. Several major themes predominate in the development. One such theme is the careful elaboration of the definition of security for a cryptosystem. A second theme has been the search for provably secure cryptosystems, based on plausible assumptions about the difficulty of specific number-theoretic problems or on the existence of certain kinds of functions (such as one-way functions). A third theme is the invention of many novel and surprising cryptographic capabilities, such as public-key cryptography, digital signatures, secret-sharing, oblivious transfers, and zero-knowledge proofs. These themes have been developed and interwoven so that today theorems of breathtaking generality and power assert the existence of cryptographic techniques capable of solving almost any imaginable cryptographic problem.

List of Sections

Index Terms for Cryptology chapter (unsorted) (Note: no names have been included in this list. Should they be?)

```
cryptology
cryptography
cryptanalysis
cryptosystem
privacy
secret-key cryptosystem
keys
encryption algorithm
decryption algorithm
cleartext
ciphertext
authentication
digital signatures
one-time-pad
Data Encryption Standard
cryptographic protocol
primality-testing
factorization
quadratic residuosity problem
modular exponentiation
discrete logarithm problem
one-way functions
passwords
message digest
fingerprint
trapdoor functions
one-way predicate
trapdoor predicate
security parameter
exponential key exchange
public-key cryptosystem
trapdoor one-way permutation
RSA
knapsack-based cryptosystems
probabilistic public-key cryptosystems
known plaintext attack
chosen ciphertext attack
polynomial-time security
multiple encryption
```

meet-in-the-middle attack
random bit sequences
pseudo-random bit sequences
pseudo-random sequence generator
linear feedback shift registers
next-bit test
pseudo-random functions
pseudo-random permutations
certificate
chosen-signature attack
forgery
existential forgery
selective forgery
universal forgery
claw-free functions
two-party protocols
user-identification
friend-or-foe
challenge-response
mental poker
commutative encryption functions
coin flipping over the telephone
oblivious transfer
contract signing
certified mail
zero-knowledge protocols
zero-knowledge proofs
graph colorability
multiparty protocols
secret sharing
verifiable secret sharing
anonymous transactions
voting
eavesdropper
adversary
cryptographic compiler
computational security
information-theoretic security
average-case complexity

Chapter 13. Cryptology

by Ronald L. Rivest[1]

1 Introduction

Ten years ago Diffie and Hellman proclaimed

> *"We stand today on the brink of a revolution in cryptography."* [53]

Today we are in the midst of that revolution. During the last decade we have seen an explosion of research in cryptology. Many cryptosystems have been proposed, and many have been broken. Our understanding of the subtle notion of "cryptographic security" has steadily increased, and cryptosystems today are routinely *proven* to be secure (after making certain plausible assumptions). The fascinating relationships between cryptology, complexity theory, and computational number theory have gradually unfolded, enriching all three areas of research.

This chapter briefly surveys the field of cryptology as it now exists, with an attempt to identify the key ideas and contributions. Lack of space precludes a comprehensive treatment, and I regret that much significant work must go unmentioned or be only briefly described.[2]

2 Basics

Cryptology is about *communication in the presence of adversaries.* As an example, a classic goal of cryptography is *privacy*: two parties wish to communicate privately, so that an adversary knows nothing about what was communicated.

The invention of radio gave a tremendous impetus to cryptography, since an adversary can eavesdrop easily over great distances. The course of World War II was significantly affected by the use, misuse, and breaking of cryptographic systems used for radio traffic [92]. It is intriguing that the computational engines designed and built by the British to crack the German *Enigma* cipher are deemed by some to be the first real "computers"; one could argue that cryptography is the mother (or at least the midwife) of computer science.

A standard cryptographic solution to the privacy problem is a *secret-key cryptosystem*, which consists of the following:

- A *message space* $\mathcal{M}$: a set of strings (*plaintext messages*) over some alphabet.
- A *ciphertext space* $\mathcal{C}$: a set of strings (*ciphertexts*) over some alphabet.

[1] This chapter prepared with support from NSF grant DCR-8607494. Author's address: MIT Lab. for Computer Science, Cambridge, Massachusetts 02139 USA. Author's net address: rivest@theory.lcs.mit.edu

[2] The reader who wishes to explore further will find available many excellent texts, collections, and survey articles [9, 13, 29, 45, 49, 48, 53, 54, 51, 66, 91, 99, 102, 117, 146, 150, 149, 148, 151], works of historical or political interest [12, 69, 92, 138, 157], relevant conference proceedings (CRYPTO, EUROCRYPT, FOCS, STOC, [46, 86, 100]) and bibliographies [14, 129].

- A *key space* $\mathcal{K}$: a set of strings (*keys*) over some alphabet.
- An *encryption algorithm* E mapping $\mathcal{K} \times \mathcal{M}$ into $\mathcal{C}$.
- A *decryption algorithm* E mapping $\mathcal{K} \times \mathcal{C}$ into $\mathcal{M}$. The algorithms E and D must have the property that $D(K, E(K, M)) = M$ for all $K \in \mathcal{K}, M \in \mathcal{M}$.

To use a secret-key cryptosystem, the parties wishing to communicate privately agree on a key K which they will keep secret (hence the name secret-key cryptosystem). They communicate a message M by transmitting the ciphertext $C = E(K, M)$. The recipient can decrypt the ciphertext to obtain the message M using K, since $M = D(K, C)$.

The cryptosystem is considered *secure* if it is infeasible in practice for an eavesdropper who learns $E(K, M)$, but who doesn't know K, to deduce M or any portion of M. This is an informal definition of security; we shall later see how this notion has been formalized, refined and improved.

As cryptography has matured it has addressed many goals other than privacy, and considered adversaries considerably more devious than a mere passive eavesdropper. One significant new goal is that of *authentication*, where the recipient of a message wishes to verify that the message he has received has not been forged or modified by an adversary and that the alleged sender actually sent the message exactly as it was received. *Digital signatures* are a special technique for achieving authentication; they are to electronic communication what handwritten signatures are to paper-based communication.

But we are getting ahead of our story. In the next two subsections we review two "pre-revolutionary" (that is, pre-1976) cryptographic techniques which are "musts" for any survey of the field: the *one-time pad* and the *Data Encryption Standard*. After that we begin our survey of "modern" cryptology.

A note on terminology: the term *cryptosystem* refers to any scheme designed to work with a communication system in the presence of adversaries, for the purpose of defeating the adversaries' intentions. This is rather broad, but then so is the field. *Cryptography* refers to the art of *designing* cryptosystems, *cryptanalysis* refers to the art of *breaking* cryptosystems, and *cryptology* is the union of cryptography and cryptanalysis.

2.1 The One-Time Pad

The *one-time pad* is a nearly perfect cryptographic solution to the privacy problem. It was invented in 1917 by Gilbert Vernam [92] for use in telegraphy and has stimulated much subsequent work in cryptography. The one-time pad is a secret-key cryptosystem where the key is as long as the message being encrypted. Furthermore, the key, once used, is discarded and never reused.

Suppose parties A and B wish to communicate privately using the one-time pad, and suppose further that they have previously agreed upon a secret key K which is a string of n randomly chosen bits. Then if A wishes to send a n-bit message M to B, she sends to B the ciphertext $C = M \oplus K$, where C is the bit-wise exclusive-or (mod-2 sum) of M and K. (For example, $0011 \oplus 0101 = 0110$.) The received ciphertext can be decrypted by B to obtain M, since $M = C \oplus K$. When another message is to be sent, another key K must be used — hence the name "one-time pad".

Russian spies have allegedly been captured with unused paper pads of printed key material for use in a one-time pad scheme. After a message was encrypted for transmission the spy would destroy the page of key material used to encrypt the message. The spy could then relax, since even if the ciphertext was intercepted it is *provably* impossible for the interceptor to decrypt the ciphertext and incriminate the spy. The proof is simple: the intercepted ciphertext C provides the interceptor no information whatsoever about M since *any* message M could have yielded C (if the key K is equal to $C \oplus M$).

This one-time pad is thus *provably* secure in an *information-theoretic* sense since the interceptor never has enough information to decrypt the ciphertext, and no amount of computational power could help him. The information-theoretic basis for cryptographic security was developed by Shannon [146] and later refined by Hellman [90]. Gilbert, MacWilliams, and Sloane [71] have also extended information-theoretic approaches to handle the *authentication* problem.

While the one-time pad provides provable security, it is awkward to use since a large key must be generated, shared, and stored. As a consequence, the one-time pad is rarely used.

2.2 The Data Encryption Standard (DES)

This subsection describes the Data Encryption Standard, our second "pre-revolutionary" cryptosystem, and one which is secure (if it is secure at all) for computational rather than information-theoretic reasons. Modern cryptosystems use relatively short keys (56 to 1000 bits), and are secure in a *computational* sense rather than in an information-theoretic sense. By this we mean that the adversary's task is *computationally infeasible* rather than information-theoretically impossible. The Data Encryption Standard (or DES) is a good example of a secret-key cryptosystem designed to be computationally secure. Researchers at IBM designed DES in the 1970's, and the U.S. National Bureau of Standards adopted DES as a standard for encrypting commercial and government unclassified information [124]. It is widely used, particularly in the banking industry. However, its future as a standard is unclear since it is not certain for how much longer NBS will support DES as a standard.

We now sketch the operation of DES. The DES algorithm takes as input a 64-bit message M and a 56-bit key K, and produces a 64-bit ciphertext C. DES first applies an initial fixed bit-permutation IP to M to obtain M'. This permutation has no apparent cryptographic significance. Second, DES divides M' into a 32-bit left half L_0 and a 32-bit right half R_0. Third, DES executes the following operations for $i = 1, \ldots, 16$ (there are 16 "rounds"):

$$L_i = R_{i-1} \tag{1}$$

$$R_i = L_{i-1} \oplus f(R_{i-1}, K_{i-1}) \tag{2}$$

Here f is a function that takes a 32-bit right half and a 48-bit *round key* and produces a 32-bit output. The function f is defined using eight substitution functions or *S-boxes*, each of which maps a 6-bit input into a 4-bit output. Each round key K_i contains a different subset of the 56 key bits. Finally, the *pre-ciphertext* $C' = (R_{16}, L_{16})$ (note that the halves are swapped) is permuted according to IP^{-1} to obtain the final ciphertext C. It is easy to verify that DES is invertible from the above definition, independent of the definition of f.

In a typical application of DES, a message M is encrypted by breaking it into 64-bit blocks, and then encrypting each block *after XOR-ing it with the ciphertext for the previous block*. This is known as *cipher-block chaining*; it prevents repeated text in the message from yielding correspondingly repeated sections of ciphertext. Other such *modes of operation* for the use of DES, as well as proposed techniques for key management, have been published by the National Bureau of Standards.

Diffie and Hellman [50] argue that the choice of a 56-bit key makes DES vulnerable to a brute force attack. For $20 million one might be able to build a machine consisting of 2^{20} chips, each of which can test 2^{20} keys/second, so that in 2^{16} seconds (18.2 hours) the entire key space can be searched for the key which maps a given plaintext into a given ciphertext.

Using a known-plaintext attack, Hellman shows how to break DES by performing a large precomputation which essentially searches the entire key space, and which saves selected results, so that a *time-memory tradeoff* results for the problem of later determining an unknown key used to encrypt the known plaintext [89].

3 The Goals and Tools of Cryptology

As cryptology has developed, the number of goals addressed has expanded, as has the number of tools available for achieving those goals. In this section we survey some key goals and tools.

Cryptology provides methods that enable a communicating party to develop trust that his communications have the desired properties, in spite of the best efforts of an untrusted party (or adversary). The desired properties may include:

- *Privacy.* An adversary learns nothing useful about the message sent.
- *Authentication.* The recipient of a message can *convince himself* that the message as received originated with the alleged sender.
- *Signatures.* The recipient of a message can *convince a third party* that the message as received originated with the alleged signer.
- *Minimality.* Nothing is communicated to other parties except that which is specifically desired to be communicated.
- *Simultaneous exchange.* Something of value (e.g., a signature on a contract) is not released until something else of value (e.g., the other party's signature) is received.
- *Coordination.* In a multi-party communication, the parties are able to coordinate their activities toward a common goal even in the presence of adversaries.
- *Collaboration threshold.* In a multi-party situation, the desired properties hold as long as the number of adversaries doesn't exceed a given threshold.

At a high level, the tools available for the attainment of these goals include:

- *Randomness.* Each party may use a private natural source of randomness (such as a noise diode) to produce "truly random" bits in order to generate his own secret keys or to perform randomized computations [72].

- *Physical protection.* Each party must physically protect his secrets from the adversary. His most important secret is usually the secret key that he has randomly generated—this key will provide him with unique capabilities.

  By contrast, design information such as equipment blueprints or cryptographic algorithm details is usually assumed to be unprotectable, so security does not usually require the secrecy of such design information. (Kerckhoffs second requirement [92, p. 235] of a cryptosystem was that "compromise of the system should not inconvenience the correspondents.")

- *Channel properties.* Unusual properties of the communication channel can sometimes be exploited. For example, Alpern and Schneider [7] show how to communicate securely on channels for which an eavesdropper cannot tell *who* broadcasts each bit. Wyner [160] defeats eavesdroppers for whom reception is less reliable than for the intended receiver, or when the channel is analog rather than digital [158, 159]. Bennett et al. exploit the peculiarities of quantum effects in their channels [17]. And spread-spectrum channels are effectively unobservable to enemies who don't know the details of their use [70]. We do not pursue these variations further in this paper.

- *Information theory.* Some systems, such as the Vernam one-time pad [92] are secure in an information-theoretic sense: the adversary is never given enough information to work with to break the code; no amount of computational power can help him overcome this. (See Shannon [146] and Hellman [90].)

- *Computational complexity theory.* The adversary's task is more often *computationally infeasible*, rather than information-theoretically impossible. Modern cryptography uses computational complexity theory to design systems that one has reason to believe cannot be broken with any reasonable amount of computation in practice, even though they are breakable in principle (with extraordinary luck—by guessing a secret key—or by using inordinate amounts of computation).

- *Cryptographic operators.* These computational mappings—such as encryption and decryption functions, one-way functions, and pseudo-random sequence generators—are basic building blocks for constructing cryptographic systems. Note that these need not be *functions*, since they may use *randomization*, so that different computations may yield different outputs, even for the same input. Complex operators may be created by composing simpler ones.

- *Cryptographic protocols.* A *protocol* specifies how each party is to initiate and respond to messages, including erroneous or illegal messages. The protocol may also specify initialization requirements, such as setting up a directory of public keys. A party following the protocol will be protected against certain specified dangers, even if the other parties do not follow the protocol.

The design of protocols and the design of operators are rather independent, in the same sense that the implemention of an abstract data type may be independent of its use. The protocol designer creates protocols assuming the existence of operators with certain security properties. The operator designer proposes implementations of those operators, and tries to prove that the proposed operators have the desired properties.

4 Mathematical Preliminaries

Many recently proposed cryptographic techniques depend heavily on number-theoretic concepts. In this section we review some basic number-theoretic and computational facts. For a more extensive review of elementary number theory see [121], [105], or [8]. An excellent overview of the problems of factoring integers, testing primality, and computing discrete logarithms also appears in this volume [103].

It is apparently the case that it is dramatically easier to tell whether a given number is prime or composite than it is to factor a given composite number into its constituent prime factors; this difference in computational difficulty is the basis for many cryptosystems. Finding large prime numbers is useful for constructing cryptographic operators, while for many cryptosystems the adversary's task is provably as hard as factoring the product of two large prime numbers.

There are efficient algorithms for generating random k-bit prime numbers; these algorithms run in time polynomial in k [152, 132, 77, 5, 3] and come in two flavors: Monte Carlo probabilistic algorithms which may err with small probability, always terminate in polynomial time and are quite efficient in practice; and Las Vegas probabilistic algorithms which are always correct, generate as output a deterministic polynomial time checkable proof of correctness, and run in expected polynomial time. Not only can random primes be generated, but Bach [11] has also shown how to create a random k-bit composite number in a factored form which which uses primality testing as a subroutine.

On the other hand, to factor a number n seems to require time proportional to

$$e^{c\cdot\sqrt{\ln(n)\cdot\ln\ln(n)}}, \tag{3}$$

where the constant c is 1 for the fastest algorithms. Factoring numbers of more than 110 decimal digits is currently infeasible in general. Pomerance [127], Pomerance et al. [128], Riesel [135], and Dixon [55] discuss recent factoring methods.

Let Z_n denote the set of residue classes modulo n, and let Z_n^* denote the multiplicative subgroup of Z_n consisting of those residues which are relatively prime to n. We let $\phi(n) = |Z_n^*|$; this is called *Euler's totient function*. Let Q_n denote the set of all *quadratic residues* (or *squares*) modulo n; that is, $x \in Q_n$ iff there exists a y such that $x \equiv y^2 \pmod{n}$.

The Jacobi symbol $(\frac{x}{n})$ is defined for any $x \in Z_n^*$ and has a value in $\{-1, 1\}$; this value is easily computed by using the law of quadratic reciprocity, even if the factorization of n is unknown. If n is prime then $x \in Q_n \Leftrightarrow (\frac{x}{n}) = 1$; and if n is composite, $x \in Q_n \Rightarrow (\frac{x}{n}) = 1$. We let J_n denote the set $\{x \mid x \in Z_n^* \wedge (\frac{x}{n}) = 1\}$, and we let $\tilde{Q}_n$ denote the set of *pseudo-squares* modulo n: those elements of J_n which do *not* belong to Q_n. If n is the product of two primes then $|Q_n| = |\tilde{Q}_n|$, and for any pseudo-square y the function $f_y(x) = y \cdot x$ maps Q_n one-to-one onto $\tilde{Q}_n$.

The *quadratic residuousity problem* is: given a composite n and $x \in J_n$, to determine whether x is a square or a pseudo-square modulo n. This problem is believed to be computationally difficult, and is the basis for a number of cryptosystems.

Squaring and extracting square roots modulo n are frequently used operations in the design of cryptographic operators. We say that x is a *square root of y, modulo n* if $x^2 \equiv y \pmod{n}$. If n has t prime factors, then x may have up to 2^t square roots. Rabin [130] proved that finding square roots modulo n is polynomial-time equivalent to factoring n; given an efficient algorithm for extracting square roots modulo n one can construct an efficient algorithm for factoring n, and vice versa. The following fact observed by Williams and Blum is also frequently useful: if n is the product of two primes each congruent to 3 mod 4, then squaring modulo n effects a permutation of Q_n.

It is frequently useful to use exponents larger than 2: the function $x^e \bmod n$ is called *modular exponentiation*; the modulus n may be either prime or composite. Unlike the squaring operator, modular exponentiation is one-to-one over Z_n if $\gcd(e, \phi(n)) = 1$ [137].

There are two ways to devise problems inverses modular exponentiation, depending on whether e or x is to be solved for. In the first case, given x, y, and n, to compute an e (if any) such that $x^e \equiv y \pmod{n}$ is called computing the *discrete logarithm* of y, modulo n (with logarithm base x). We denote such an e as $index_{x,n}(y)$. We note that when n is prime there are many x such that $x^e \equiv y \pmod{n}$ has solutions for all $y \in Z_n^*$; such x's are called *generators*. Computing discrete logarithms seems to be difficult in general. However, Pohlig and Hellman [126] present effective techniques for this problem when n is prime and $n - 1$ has only small prime factors. Adleman [2] shows how to compute discrete logarithms in the time given in equation (3), so that computing discrete logarithms and factoring integers seem to have essentially the same difficulty, as a function of the size of the numbers involved. It interesting to note that working over the finite field $GF(2^k)$ rather than working modulo n seems to make the problem substantially easier (see Coppersmith [43] and Odlyzko [123]). See [44] for further improvements and general discussion.

The other way to invert modular exponentiation is to solve for x: given y, e, and n, to compute an x (if any) such that $x^e \equiv y \pmod{n}$ is called computing the *e-th root of y modulo n*. If n is prime, this computation can be performed in polynomial time [22, 4, 133], while if n is composite, this problem seems to be as hard as factoring n or computing discrete logarithms modulo n.

We say that a binary random variable X is *ϵ-biased* if the probability that $X = 0$ is within ϵ of 1/2: that is, if $P(X = 0) \in [1/2 - \epsilon, 1/2 + \epsilon]$. This notion will be useful in our discussion of pseudo-random bit sequences.

5 Complexity-Theoretic Foundations of Cryptography

Modern cryptography is founded on computational complexity theory. When we say that a system has been proven secure, we mean that a lower bound has been proved on the number of computational steps required to break the system. At this time, however, the young field of complexity theory has yet to prove a non-linear lower bound for even one NP-complete problem. Thus, the theory of cryptography today is based on certain unproved but seemingly plausible assumptions about the computational difficulty of solving certain problems, and the assumed existence of operators like one-way functions and trapdoor functions. In this section we review these operators.

5.1 Checksums and One-Way Functions

It is often useful to work with a function f which can take as input a long message M and produce as output a shorter value $f(M)$. Depending on the application, the function f we choose may need to have different properties. Typically, verifying the correspondence between M and $f(M)$ allows one to verify, with high confidence, that the message M has not been altered since $f(M)$ was computed.

The simplest such f are *checksums*. For example, a simple checksum of M is obtained by interpreting the bits of M as coefficients of a polynomial $M(x)$ over $GF(2)$ and taking $f(M)$ as the remainder when $M(x)$ is divided by a fixed polynomial $p(x)$. Cyclic redundancy checksums are of this type. If the pair $(M, f(M))$ is transmitted over a noisy channel, transmission errors can be detected when the received pair (x, y) doesn't satisfy $y = f(x)$. Such a strategy works well against random errors, but not against malicious tampering by an adversary. Since anyone can compute f, this procedure provides the recipient no authentication regarding the identity of the sender. Checksums are therefore *not* suitable for typical cryptographic applications.

A more useful function for cryptographic applications is a *one-way function*. Such a function takes a message M and efficiently produces a value $f(M)$ such that it is computationally infeasible for an adversary, given $f(M) = z$, to find *any* message M' whatsoever (including $M' = M$) such that $f(M') = z$. A slightly stronger requirement is that it should be computationally infeasible for the adversary, given f, to come up with any pair of messages (x, y) such that $f(x) = f(y)$; such a function we call *claw-free*. (Because of the *birthday paradox*, for small message spaces it may be feasible in practice to find such a pair, even though it is infeasbile in practice to invert f at a given point z [164].)

A publicly-available one-way function has a number of useful applications.

1. In a time-shared computer system, instead of storing a table of login passwords, one can store, for each password w, the value $f(w)$. Passwords can easily be checked for correctness at login, but even the system administrator can not deduce any user's password by examining the stored table [59].

2. In a public-key cryptosystem (see the following sections), it may be more efficient to sign $f(M)$ rather than signing M itself, since M may be relatively long whereas f can be designed to return a fixed-length result. Thus, using $f(M)$ keeps the size of a signature bounded. In this application $f(M)$ is sometimes called a *message digest* or *fingerprint* for the message M. Since no one can come up with two messages that have the same digest, the signer is protected against an adversary altering his signed messages.

5.2 Trapdoor Functions

A *trapdoor function* f is like a one-way function except that there also exists a secret inverse function f^{-1} (the *trapdoor*) that allows its possessor to efficiently invert f at any point of his choosing. It should be easy to compute f on any point, but infeasible to invert f on any point without knowledge of the inverse function f^{-1}. Moreover, it should be easy to generate matched pairs of f's and corresponding f^{-1}'s. One such a matched pair is generated, the publication of f should not reveal anything about how to compute f^{-1} on any point.

Trapdoor functions have many applications, as we shall see in the next sections. They are the basis for public-key cryptography. The ability to invert a particular trapdoor function f will uniquely identify a given party. That is, each party will publish his own trapdoor function f, and only the party who publishes a given trapdoor function f will be able to invert that function.

5.3 One-Way (and Trapdoor) Predicates

A *one-way predicate*, first introduced in [79, 78], is a boolean function $B : \{0,1\}^* \rightarrow \{0,1\}$ such that

1. On input $v \in \{0,1\}$ and 1^k, in expected polynomial time one can choose an x such that $B(x) = v$ and $|x| \leq k$, randomly and uniformly from the set of all such x.

2. For all $c > 0$, for all k sufficiently large, no polynomial-time adversary given $x \in \{0,1\}^*$ such that $|x| \leq k$ can compute $B(x)$ with probability greater than $\frac{1}{2} + \frac{1}{k^c}$. (The probability is taken over the random choices made by the adversary and x such that $|x| \leq k$.)

A *trapdoor predicate* is a one-way predicate for which there exists, for every k, trapdoor information t_k whose size is bounded by a polynomial in k and whose knowledge enables the polynomial-time computation of $B(x)$, for all x such that $|x| \leq k$.

These primitives are the basis for the probabilistic constructions for privacy and pseudo-random number generation discussed in later sections. Each party publishes his own trapdoor predicate B, and keeps private the associated trapdoor information which enables him alone to compute $B(x)$.

5.4 Making Appropriate Complexity-Theoretic Assumptions

Computational complexity theory works primarily with asymptotic complexity—what happens as the size of the problem becomes large. In order to apply notions of computational complexity theory to cryptography we must typically envisage not a single cryptosystem or cryptographic function but a family of them, parameterized by a *security parameter* k. That is, for each value of the security parameter k there is be a specific cryptosystem or function. Or, there may be a family of cryptosystems or functions for each value of k. We can imagine that a cryptosystem with security parameter k has inputs, outputs, and keys which are all strings of length k (or some suitable polynomial function of k). As the security parameter k becomes large, the complexity of the underlying mathematical problems embedded in the cryptosystem should become sufficiently great that one can hope that cryptographic security is obtained. Since we don't know for sure that $P \neq NP$, a "proof" of security is necessarily dependent on the assumption that certain computational problems are difficult as the inputs become large.

As an example, the assumption that factoring integers is hard might be formalized as: for any probabilistic polynomial-time (factoring) algorithm A, for all constants $c > 0$ and sufficiently large k, the chance that A can produce a nontrivial divisor of its input (where the input is the product of two randomly chosen k-bit primes) is at most $1/k^c$. Note that A may be a probabilistic algorithm, since any adversary worth his salt can also flip coins.

Then a careful proof of security would show that the ability of the adversary to defeat the cryptographic system a significant fraction of the time would contradict the assumed difficulty of the hard problem (e.g., factoring). This generally takes the form of a reduction, where it is shown how to solve the hard problem, given the ability to break the cryptographic system.

When formally defining the above primitives (one-way and trapdoor functions and predicates) one must carefully choose what computational power to give to the adversary attempting to break (i.e., invert) the primitive. The computational model should be strong enough to capture the computational power of real-life adversaries. To this end, we let the polynomial-time adversary be not only probabilistic but also non-uniform. The latter is appropriate since cryptosystems are often designed with a fixed size (that is, security parameter) in mind. Moreover, the most meaningful proofs of security are necessarily those proved with respect to the most powerful adversary. Thus, the adversary is modeled as an infinite family of probabilistic circuits (one for every security parameter k), whose size grows as a polynomial in k. However, the extent to which allowing an adversary to be non-uniform is really meaningful remains to be seen, since making this assumption normally just means that the assumption that (say) factoring is difficult, when formalized, has to be reformulated to say that factoring is even difficult for a non-uniform factoring procedure.

6 Privacy

The goal of privacy is to ensure that an adversary who overhears a given transmission learns nothing useful about the message transmitted. There are several ways of achieving this goal, which are sketched in this section.

6.1 Secret-key cryptosystems

A simple method to achieve privacy is to use a conventional secret-key cryptosystem such as DES. The parties wishing to communicate must initally arrange to share a common secret key K; this key is then used to encrypt all messages transmitted. An adversary who does not possess the shared secret key will not be able to decrypt any encrypted messages he happens to overhear. The difficulty with secret-key cryptosystems is that it is often awkward to establish the initial distribution of the secret shared key, a problem we now discuss.

A simple solution to the initialization problem is to use a courier to distribute the keys. This method requires that the courier visit each party who must be given the secret key. The courier must be trusted to ensure that the key is only given to the appropriate parties.

Sometimes it is known (or assumed) a priori that an adversary will be passive—that is, the adversary may eavesdrop on transmitted messages but will not transmit any messages himself. In such a case two parties can establish a shared secret key using *exponential key exchange*; an elegant technique proposed by Diffie and Hellman [53]. Let A and B be the two parties, and suppose that A and B agree (via a public dialogue that anyone can overhear) on a large prime p and a generator g of the multiplicative group Z_p^*. Then A and B choose respective large secret integers a and b, and they exchange with each other the values $g^a \bmod p$ and $g^b \bmod p$. Now A can compute $g^{ab} \bmod p$ (from a and g^b), and B can compute the same value (from b and g^a). Thus A and B can use the value $g^{ab} \bmod p$ as their shared secret key. An adversary who wants to determine this key is left with the problem of computing $g^{ab} \bmod p$ from $g^a \bmod p$ and $g^b \bmod p$; an apparently intractable problem.

6.2 Deterministic Public-Key Encryption

The notion of a *public-key cryptosystem* was first published by Diffie and Hellman in 1976 [52, 53], although Merkle had earlier developed some of the conceptual framework [115]. The central idea is that of a trapdoor function, as defined earlier. According to Diffie and Hellman, a public-key cryptosystem should contain the following parts:

- A key generation algorithm. This is a randomized polynomial-time algorithm $\mathcal{G}$, which, on input k (the security parameter) produces a public key E, and a corresponding private key D.
- An encryption algorithm. This algorithm takes as input a public key E and a message M, produces a ciphertext C, which we denote $E(M)$. This operation is one-to-one.
- A decryption algorithm. This algorithm takes as input a private key D and a ciphertext C, and produces a corresponding message $M = D(C)$.

These algorithms have the following properties:

- For every message M, $D(E(M)) = M$.
- For every message M, $E(D(M)) = M$.
- The key-generation, encryption, and decryption algorithms run in time polynomial in the length of their inputs. The key-generation algorithm is a randomized algorithm; the encryption and decryption algorithms are deterministic.
- Given the public key E, but not the private key D, the chance that a polynomial-time adversary can decrypt a random ciphertext $C = E(M)$ is less than any polynomial fraction. (Here M is chosen at random from the set of all messages, and we may take "polynomial fraction" to mean at least k^{-c} for some constant c and all sufficiently large k.) This implies that E is a trapdoor function.

We might also call the encryption algorithm a *trapdoor one-way permutation*, since it provides a permutation of the message space and since you can only go one way (from message to ciphertext but not vice-versa) without knowledge of the secret trapdoor information D.

If user A of a communication network publishes her public key E_A in a directory of public keys, then anyone can send A private mail by encrypting a message M with A's public key. Only A possesses the decryption key D_A, so only A can decrypt such a message.

6.2.1 RSA

In 1977 Rivest, Shamir, and Adleman [137] proposed a public-key cryptosystem satisfying the requirements proposed by Diffie and Hellman. In their scheme, each user's public key is a pair (e, n) of integers, such that n is the product of two large primes p and q and $\gcd(e, \phi(n)) = 1$. The encryption operation is then

$$C = M^e \pmod{n}. \tag{4}$$

The corresponding private key is the pair (d, n), where $d \cdot e \equiv 1 \pmod{\phi(n)}$, and the decryption operation is

$$M = C^d \pmod{n}. \tag{5}$$

There are several reasons to believe that RSA is secure. For example, it is provably as hard to derive the private key from the public key as it is to factor n. Furthermore, the RSA system is *multiplicative*: $E(X) \cdot E(Y) = E(X \cdot Y)$. For this reason, if an adversary could decrypt any polynomial fraction of the ciphertexts in polynomial time, then he could decrypt all ciphertexts in random polynomial time: to decrypt $E(X)$ it suffices to find (by random trial and error) a Y such that $E(X \cdot Y)$ (which is the same as $E(X)E(Y)$) can be decrypted (yielding XY), and then dividing the result by Y to obtain X. One might interpret this as saying that either RSA is uniformly secure or it is uniformly insecure.

Even stronger results have been proven. For example, it has been shown [81, 6, 18] that if a polynomial fraction of RSA ciphertexts can't be decrypted in polynomial time, then neither can just the least significant bit of the message be guessed from the ciphertext with better than an ϵ bias.

Hastad [88] shows that it is unwise to use a low encryption exponent e, such as 3, if it is likely that a user may send the same message (or the same message with known variations) to a number of other users.

6.2.2 Knapsacks

A number of public-key cryptosystems have been proposed which are based on the *knapsack* (or — more properly — the *subset sum*) problem: given a vector $\mathbf{a} = (a_1, a_2, \ldots, a_n)$ of integers, and a target value C, to determine if there is a length-n vector $\mathbf{x}$ of zeroes and ones such that $\mathbf{a} \cdot \mathbf{x} = C$. This problem is NP-complete [68].

To use the knapsack problem as the basis for a public-key cryptosystem, you create a public key by creating a knapsack vector $\mathbf{a}$, and publish that as your public key. Someone else can send you the encryption of a message M (where M is a length-n bit vector), by sending you the value of the inner product $C = M \cdot \mathbf{a}$. Clearly, to decrypt this ciphertext is an instance of the knapsack problem. To make this problem easy for you, you need to build in hidden structure (that is, a trapdoor) into the knapsack so that the encryption operation becomes one-to-one and so that you can decrypt a received ciphertext easily. It seems, however, that the problem of solving knapsacks containing a trapdoor is *not* NP-complete, so that the difficulty of breaking such a knapsack is no longer related to the $P = NP$ question.

In fact, history has not been kind to knapsack schemes; most of them have been broken by extremely clever analysis and the use of the powerful L^3 algorithm [104] for working in lattices. See [114, 140, 142, 1, 144, 100, 32, 122].

Some knapsack or knapsack-like schemes are still unbroken. The Chor-Rivest scheme [39], and the multiplicative versions of the knapsack [114] are examples. McEliece has a knapsack-like public-key cryptosystem based on error-correcting codes [113]. This scheme has not been broken, and was the first scheme to use randomization in the encryption process.

6.3 Probabilistic Public-Key Encryption

With the introduction of randomized or probabilistic cryptographic techniques, it becomes possible to propose satisfactory definitions of the mathematical security of a cryptographic system, and to prove that certain cryptosystems are secure under this definition (under suitable complexity-theoretic assumptions, as usual).

These probabilistic cryptosystems will make use of the one-way function and trapdoor functions primitives in a much more complex fashion than their earlier deterministic counterparts which (although quite useful and secure in practice) will not be able to satisfy the security definitions given here.

The pioneering work in this direction was performed by Goldwasser and Micali [79, 78]. Although the use of randomized techniques was itself not new (for example, [113]), using randomization to achieve a *provable* level of security was novel.

We begin by examining the rather subtle notion of cryptographic security.

6.3.1 Attacks Against a Cryptosystem

When proving that a cryptographic system is secure, it is important to carefully specify what sort of "attacks" the adversary may mount. It is not uncommon for a system to be secure against a weak attack (such as passive eavesdropping) but to be insecure against a more powerful attack (such as active eavesdropping and manipulation of the communication line).

In the simplest form of attack, the passive adversary merely observes legitimate users using the cryptographic system. He may see ciphertext only, or he may be able to see some plaintext/ciphertext pairs (a *known plaintext* attack). In a public key setup he can always generate a polynomial number of pairs of plaintext/ciphertext himself, using the public key.

A potentially more powerful attack, which has been considered and is probably more realistic in practice, is that of an adversary who is a legitimate user of the system himself. The adversary is then able to perform all of the actions permitted by a legitimate user before trying to break (decipher) ciphertexts sent between pairs of users.

More generally, we might assume that an adversary can manipulate the communications between any pair of legitimate users, and can even temporarily run their cryptographic equipment (but can't take it apart to see its secret keys). For example, in a *chosen ciphertext* attack the adversary is assumed to be able to see the plaintext corresponding to ciphertexts of his choice.

6.3.2 The Goals of the Adversary

"Success" for the adversary should be defined in the most generous manner, so that a proof of security rules out even the weakest form of success.

A modest goal to aim for in the design of a cryptosystem is that for most messages the adversary can not derive the entire message from its ciphertext. This is too modest a goal, since it does not preclude the following problems:

- The cryptosystem is not secure for some probability distributions on the message space (e.g., the message space consists of only a polynomial number of messages which are known to the adversary).
- Partial information about messages may be easily computed from the cipher text.
- It may be easy to detect simple but useful facts about traffic of messages, such as when the same messages is sent more than once.

Note that deterministic public key cryptosystems as proposed by Diffie and Hellman achieve the above modest goal and yet suffer from the above problems:

- If m is drawn from a highly structured sparse message space, then $f(m)$ may be easy to invert (e.g., for the RSA function $f(0)$ and $f(1)$ are easily detectable).
- Some information about m is always easy to compute from $f(m)$ for any f, such as "the parity of $f(m)$" (or worse: for one-way candidate $f(x) = g^x \bmod p$ where p is prime and g is a generator, the least significant bit of x is easily computable from $f(x)$).

A much more ambitious goal is to require that for all probability distributions over the message spaces, the adversary can not predict from the ciphertext with significantly increased accuracy any bit of information about the corresponding messages.

Essentially, this coincides with the existing formal security notions. Achieving it rules out all of the problems listed above.

6.3.3 Definition of Security: Polynomial-Time Security

Several definitions of security for probabilistic encryption schemes have been proposed and studied in [79, 163, 78]. All definitions proposed so far have been shown to be equivalent in [78, 118]. We provide one definition in detail, due to Goldwasser and Micali [78].

Definition: We say that a *probabilistic encryption scheme is polynomial-time secure* if for all sufficiently large security parameters k, any probabilistic polynomial-time procedure that takes as input k (in unary) and a public key, and that produces as output two messages m_0 and m_1, can not distinguish between a random encryption of m_0 and a random encryption of m_1 with probability greater than $1/2 + 1/k^c$ for all c.

6.3.4 Probabilistic Encryption

We begin by describing the probabilistic encryption technique proposed in [79, 78].

Alice creates a public key consisting of two parts: an integer n which is the product of two large primes p and q, and a pseudo-square $y \in \tilde{Q}_n$. We assume that the quadratic residuousity problem is hard, so that Alice (who knows the factorization of n) can distinguish squares from pseudo-squares modulo n, but Bob (who knows only n) can not decide in probabilistic polynomial time whether x in J_n is a square or not mod n. The following theorem due to Goldwasser and Micali shows that if this decision problem is hard at all then it is everywhere hard.

Theorem 1 (Goldwasser-Micali[79, 78]) *Let $S \subset \{n|n = pq, |p| \approx |q|\}$. If there exists a probabilistic polynomial-time algorithm A such that for $n \in S$,*

$$P(A(n,x) \text{ decides correctly whether } x \in Q_n | x \in J_n) > \frac{1}{2} + \epsilon \ , \tag{6}$$

where this probability is taken over the choice of $x \in J_n$ and A's random choices, then there exists a probabilistic algorithm B with running time polynomial in $\epsilon^{-1}, \delta^{-1}$ and $|n|$ such that for all $n \in S$, for all $x \in J_n$,

$$P(B(x,n) \text{ decides correctly whether } x \in Q_n | x \in J_n) > 1 - \delta \ , \tag{7}$$

where this probability is taken over the random coin tosses of B.

Namely, a probabilistic polynomial-time bounded adversary can not do better (except by a smaller than any polynomial advantage) than guess at random whether $x \in J_n$ is a square mod n, if quadratic residuosity problem is not in BPP.

To send a message M to Alice using a probabilistic encryption scheme, Bob proceeds as follows. Let $M = m_1 m_1 \ldots m_k$ in binary notation. For $i = 1, \ldots, k$, Bob:

1. Randomly chooses an integer r from Z_n.
2. If $m_i = 0$, sends $c_i = r^2 \bmod n$ to Alice; if $m_i = 1$, sends $c_i = y \cdot r^2 \bmod n$ to Alice.

When $m_i = 0$, Bob is sending a random square to Alice, whereas if $m_i = 1$, B is sending a random pseudo-square. (Alice needs to include y in her public key just so that Bob will be able to generate random pseudo-squares.) Since Alice can distinguish squares from pseudo-squares modulo n, she can decode the message.

Although decryption is easy for Alice, for an adversary the problem of distinguishing whether a given piece of ciphertext c_i represents a 0 or a 1 is *precisely* the problem of distinguishing squares from pseudo-squares that was assumed to be hard.

A natural generalization of the scheme based on any trapdoor predicate follows from [78]. Recall, that a trapdoor predicate is a boolean function $B : \{0,1\}^* \rightarrow \{0,1\}$ such that it is easy in expected polynomial time on input 1^k and bit v to choose an x randomly such that $B(x) = v$ and $|x| \leq k$; and no polynomial-time adversary given random $x \in X$ such that $|x| \leq k$ can compute $B(x)$ with probability greater than $1/2 + 1/k^c$, for all $c > 0$ and for all sufficiently large k; if the trapdoor information is known however, it is easy to compute $B(x)$.

Fix a trapdoor predicate B. Let the security parameter of the system be k. Alice's public key contains a description of B, and her secret key contains the trapdoor information. Now Bob can send Alice a private message $M = m_1 m_1 \ldots m_k$ (this is M in binary notation) as follows. For $i = 1, \ldots, k$, Bob:

1. Randomly chooses a binary string x_i of length at most k such that $B(x) = m_i$.

2. Sends x_i to Alice.

To decrypt, Alice, who knows the trapdoor information, computes $m_i = B(x_i)$ for all $i = 1, ..., k$.

Theorem 2 (Goldwasser-Micali[78]) *If trapdoor predicates exist, then the above probabilistic public-key encryption scheme is polynomial-time secure.*

Implementation of trapdoor predicates based on the problem of factoring integers, and of inverting the RSA function can be found in [6]. We outline the RSA-based implementation. Let n be the public modulus, e the public exponent, and d the secret exponent. Let $B(x)$ be the least significant bit of $x^d \bmod n$ for $x \in Z_n^*$. Then, to select uniformly an $x \in Z_n^*$ such that $B(x) = v$ simply select a $y \in Z_n^*$ whose least significant bit is v and set $x = y^e \bmod n$.

Yao, in a pioneering paper [163], showed that the existence of any trapdoor length-preserving permutation implies the existence of a trapdoor predicate. Recently, Goldreich and Levin simplified Yao's construction as follows.

Theorem 3 (Goldreich-Levin[75]) *If trapdoor length-preserving permutations exist, then B is a trapdoor predicate, where B is defined as follows. Let $f : \{0,1\}^* \rightarrow \{0,1\}^*$ be a trapdoor length-preserving permutation. Let $B(f(x), y) = xy \bmod 2$ (the inner-product of x and y).*

Now, to encrypt a single bit v, Alice simply selects at random a pair x, y of values from $\{0,1\}^k$ such that $xy \bmod 2 = v$ and obtains the ciphertext as $c = f(x)y$, the concatentation of $f(x)$ and y.

How efficent are the probabilistic schemes? In the schemes described so far, the ciphertext is longer than the cleartext by a factor proportional to the security parameter. However, it has been shown [23, 27] using later ideas on pseudo-random number generation how to start with trapdoor functions and build a probabilistic encryption scheme that is polynomial-time secure for which the ciphertext is longer than the cleartext by only an additive factor. The most efficient probabilistic encryption scheme is due to Blum and Goldwasser [27] and is comparable with the RSA deterministic encryption scheme in speed and data expansion.

6.4 Composition of Cryptographic Operators and Multiple Encryption

Sometimes new cryptographic operators can be obtained by composing existing operators. The simplest example is that of multiple encryption.

Does multiple encryption increase security? Not always: consider the class of *simple substitution ciphers*, where the plaintext is turned into the ciphertext by replacing each plaintext letter with the corresponding ciphertext letter, determined according to some table. (Newspaper cryptograms are usually of the sort.) Since composing two simple substitution ciphers yields another simple substitution cipher, multiple encryption does not increase security.

On the other hand, multiple encryption using DES probably does increase security somewhat (see [95, 147]).

It is worth noting that the composition of two cryptosystems with n-bit keys can be broken in time $O(2^n)$ and space $O(2^n)$, using a *meet-in-the-middle* attack. Given a matching plaintext/ciphertext pair for the composed system, one can make a table of size 2^n of all possible intermediate values obtainable by encrypting the plaintext with the first system, and a second table of size 2^n of all possible intermediate values obtainable by decrypting the ciphertext with the second system. By sorting the two tables (which can be done in linear time using a bucket sort), and looking for overlaps, one can identify a pair of keys that take the given plaintext to the given ciphertext. Thus the composed system has difficulty proportional to 2^n, rather than proportional to 2^{2n}, as one would naively expect.

One very intuitive result, due to Even and Goldreich [60] (see also [10]), is that the composition of encryption schemes A and B is no weaker than A or B individually—security is not lost by composition. They assume that the two encryption keys are chosen *independently* and that the adversary can request the encryption of arbitrary text (that is, he can use a *chosen-plaintext* attack).

Luby and Rackoff [112, 111] prove a more powerful result, which shows that the composition generally *increases* security. Define an encryption scheme A to be $(1-\epsilon)$*-secure* if no polynomial-time procedure has a chance greater than $(1+\epsilon)/2$ of distinguishing encryption functions from scheme A from truly random functions over the same domain (see [112] for a more precise definition and details). Then the composition of a $(1-\epsilon)$-secure encryption scheme with a $(1-\delta)$-secure encryption scheme is $(\epsilon\delta(2-\max(\epsilon,\delta)))$-secure. This is an improvement whenever $\max(\epsilon,\delta) < 1$.

In a similar vein, a number of researchers [78, 163, 106] have developed and refined proofs that the bit-wise XOR of several independent pseudo-random bit sequence generators is harder to predict (by a quantifiable amount) than any of the component generators.

7 Generating Random or Pseudo-Random Sequences and Functions

We now examine in some detail the problem of generating random and pseudo-random sequences. One motivation for generating random or pseudo-random sequences is for use in the one-time pad, as described previously.

7.1 Generating Random Bit Sequences

Generating a one-time pad (or, for that matter, any cryptographic key) requires the use of a "natural" source of random bits, such as a coin, a radioactive source or a noise diode. Such sources are absolutely essential for providing the initial secret keys for cryptographic systems.

However, many natural sources of random bits may be defective in that they produce *biased* output bits (so that the probability of a one is different than the probability of a zero), or bits which are *correlated* with each other. Fortunately, one can remedy these defects by suitably processing the output sequences produced by the natural sources.

To turn a source which supplies biased but uncorrelated bits into one which supplies unbiased uncorrelated bits, von Neumann proposed grouping the bits into pairs, and then turning 01 pairs into 0's, 10 pairs into 1's, and discarding pairs of the form 00 and 11 [156]. The result is an unbiased uncorrelated source, since the 01 and 10 pairs will have an identical probability of occurring. Elias [58] generalizes this idea to achieve an output rate near the source entropy.

Handling a correlated bit source is more difficult. Blum [26] shows how to produce unbiased uncorrelated bits from a biased correlated source which produces output bits according to a known finite Markov chain.

For a source whose correlation is more complicated, Santha and Vazirani [139] propose modelling it as a *slightly random source*, where each output bit is produced by a coin flip, but where an adversary is allowed to choose *which* coin will be flipped, from among all coins whose probability of yielding "Heads" is between δ and $1-\delta$. (Here δ is a small fixed positive quantity.) This is an extremely pessimistic view of the possible correlation; nonetheless U. Vazirani [153] shows that if one has *two, independent*, slightly-random sources X and Y then one can produce "almost independent" ϵ-biased bits by breaking the outputs of X and Y into blocks $\mathbf{x}, \mathbf{y}$ of length $k = \Omega(1/\delta^2 \log(1/\delta)\log(1/\epsilon))$ bits each, and for each pair of blocks $\mathbf{x}, \mathbf{y}$ producing as output the bit $\mathbf{x} \cdot \mathbf{y}$ (the inner product of $\mathbf{x}$ and $\mathbf{y}$ over $GF(2)$). This is a rather practical and elegant solution. Chor and Goldreich [37] generalize these results, showing how to produce independent ϵ-biased bits from even worse sources, where some output bits can even be completely determined.

These results provide effective means for generating truly random sequences of bits—an essential requirement for cryptography—from somewhat defective natural sources of random bits.

7.2 Generating Pseudo-Random Bit or Number Sequences

The one-time pad is generally impractical because of the large amount of key that must be stored. In practice, one prefers to store only a short random key, from which a long pad can be produced with a suitable cryptographic operator. Such an operator, which can take a short *random* sequence x and deterministically "expand" it into a *pseudo-random* sequence y, is called a *pseudo-random sequence generator*. Usually x is called the *seed* for the generator. The sequence y is called "pseudo-random" rather than random since not all sequences y are possible outputs; the number of possible y's is at most the number of possible seeds. Nonetheless, the intent is that for all practical purposes y should be indistinguishable from a truly random sequence of the same length.

It is important to note that the use of pseudo-random sequence generator reduces *but does not eliminate* the need for a natural source of random bits; the pseudo-random sequence generator is a "randomness expander", but it must be given a truly random seed to begin with.

To achieve a satisfactory level of cryptographic security when used in a one-time pad scheme, the output of the pseudo-random sequence generator must have the property that an adversary who has seen a portion of the generator's output y must remain unable to efficiently predict other unseen bits of y. For example, note that an adversary who knows the ciphertext C can guess a portion of

y by correctly guessing the corresponding portion of the message M, such as a standarized closing "Sincerely yours,". We would not like him thereby to be able to efficiently read other portions of M, which he could do if he could efficiently predict other bits of y. Most importantly, the adversary should not be able to efficiently infer the seed x from the knowledge of some bits of y.

How can one construct secure pseudo-random sequence generators?

7.2.1 Classical Pseudo-random Generators are Unsuitable

Classical techniques for pseudo-random number generation [98, Chapter 3] which are quite useful and effective for Monte Carlo simulations are typically unsuitable for cryptographic applications. For example, *linear* feedback shift registers [85] are well-known to be cryptographically insecure; one can solve for the feedback pattern given a small number of output bits.

Linear congruential random number generators are also insecure. These generators use the recurrence

$$X_{i+1} = aX_i + b \pmod{m} \tag{8}$$

to generate an output sequence $\{X_0, X_1, \ldots\}$ from secret parameters a, b, and m, and starting point X_0. It is possible to infer the secret parameters given just a few of the X_i [125]. Even if only a fraction of the bits of each X_i are revealed, but a, b, and m are known, Frieze, Hastad, Kannan, Lagarias, and Shamir show how to determine the seed X_0 (and thus the entire sequence) using the marvelous *lattice basis reduction* (or "L^3") algorithm of Lenstra, Lenstra, and Lovász [65, 104].

As a final example of the cryptographic unsuitability of classical methods, Kannan, Lenstra, and Lovasz [96] use the L^3 algorithm to show that the binary expansion of any algebraic number y (such as $\sqrt{5} = 10.001111000110111\ldots$) is insecure, since an adversary can identify y exactly from a sufficient number of bits, and then extrapolate y's expansion.

7.2.2 Provably Secure Pseudo-Random Generators

The first pseudo-random sequence generator proposed which was *provably secure* (assuming that it is infeasible to invert the RSA function (see section 6.2.1)) is due to Shamir [143]. However, this scheme generates a sequence of *numbers* rather than a sequence of *bits*, and the security proof shows that an adversary is unable to predict the next *number*, given the previous numbers output. This is not strong enough to prove that, when used in a one-time pad scheme, each *bit* of the message will be well-protected.

Blum and Micali [28] introduced the first method for designing provably secure pseudo-random *bit* sequence generators, based on the use of one-way predicates. Let D denote a finite domain, let $f : D \rightarrow D$ a permutation of D, and let B denote a function from D to $\{0, 1\}$ such that (i) it is easy to compute $B(y)$, given $x = f^{-1}(y)$, and (ii) it is difficult to compute $B(y)$, given only y.

Given a seed $x_0 \in D$, we can create the sequence $x_0, x_1, \ldots, x_n$ using the recurrence $x_{i+1} = f(x_i)$. To produce an output binary sequence $b_0, \ldots, b_{n-1}$ of length n, define $b_i = B(x_{n-i})$. Note the reversal of order relative to the x sequence; we must compute $x_0, \ldots, x_n$ first, and then compute b_0 from x_{n-1}, b_1 from x_{n-2}, $\ldots$, and b_{n-1} from x_0.

If a pseudo-random bit sequence generator has the property that it is difficult to predict b_{i+1} from $b_0, \ldots, b_i$ with accuracy greater than $(1+\epsilon)/2$ in time polynomial in $1/\epsilon$ and the size of the seed, then we say that the generator *passes the "next-bit" test.*

Blum and Micali prove that their generator passes the next-bit test as follows. Suppose otherwise. Then we derive a contradiction by showing how to compute $B(y)$ from y. Because f is a permutation, there is an x_0 such that $y = x_{n-i-1}$ in the sequence generated starting from x_0. Given $y = x_{n-i-1}$, we can compute $x_{n-i}, x_{n-i+1}, \ldots, x_n$ as well, using f, and thus we can compute $b_0, \ldots, b_i$ from y. If we can then predict $b_{i+1} = B(x_{n-i-1}) = B(y)$ efficiently, we have contradicted our assumption that $B(y)$ is difficult to compute from y alone. (The above proof sketch is made rigorous in [28]; the phrases "easy" and "hard" are made precise, and the definition of computing $B(y)$ from y is generalized to include being able to predict $B(y)$ with an accuracy greater than 1/2.)

Blum and Micali then proposed a particular generator based on the difficulty of computing discrete logarithms and the above method, as follows. Define $B(x) = B_{g,p}(x)$ to be 1 if $index_{g,p}(x) \leq (p-1)/2$, and 0 otherwise, where p is a prime, g is a generator of Z_p^*, and $x \in Z_p^*$, and define $f(x) = f_{g,p}(x) = g^x \bmod p$. If computing discrete logarithms modulo p is indeed difficult, then the sequences produced will be unpredictable.

Blum, Blum, and Shub [23] propose another generator, called the *$x^2 \bmod n$ generator*, which is simpler to implement and also provably secure (assuming that the quadratic residuousity problem is hard). This generator follows the Blum-Micali general method, with $B(x) = 1$ iff x is odd, and $f(x) = x^2 \bmod n$. Alexi, Chor, Goldreich, and Schnorr [6] show that the assumption that the quadratic residuosity problem is hard can be replaced by the weaker assumption that factoring is hard. A related generator is obtained by using the RSA function $x^e \bmod n$ where $\gcd(e, \phi(n)) = 1$ [154, 6]. Kaliski shows how to extend these methods so that the security of the generator depends on the difficulty of computing elliptic logarithms; his techniques also generalize to other groups [94, 93].

To improve efficiency, it is desirable to obtain as many random bits as possible from each application of f. That is, $B(x)$ should return more than one bit of information. Long and Wigderson [109] show how to extract $c \log \log p$ pseudo-random bits from each x_i instead of just one bit as in the Blum-Micali generator. A similar result has been shown for the RSA generator [6].

Yao [163] shows that the pseudo-random generators defined above are *perfect* in the sense that no probabilistic polynomial-time algorithm can guess with probability greater than $1/2 + \epsilon$ whether an input string of length k was randomly selected from $\{0,1\}^k$ or whether it was produced by one of the above generators. One can rephrase this to say that a generator that passes the next-bit test is perfect in the sense that it will *pass all polynomial-time statistical tests.* The Blum-Micali and Blum-Blum-Shub generators, together with the proof of Yao, represent a major achievement in the development of provably secure cryptosystems. Levin [106] shows that perfect pseudo-random bit generators exist if and only if there exists a one-way function f that can not be inverted easily at points of the form $f^t(x)$, the t-th iterate of f applied to a random point $x \in \{0,1\}^k$.

7.2.3 Pseudo-Random Functions and Permutations

More generally, one can imagine having a family $f_j(\cdot)$ of (pseudo-random) functions. The index j can be thought of as the *key* selecting which function is in use.

Such a family of functions can be used for authentication. If two users share a secret key j, then they can authenticate their messages to each other by appending the tag $f_j(M)$ to a message M. It should be infeasible for an adversary, seeing $f_j(M_1), \ldots, f_j(M_n)$, to produce a valid tag $f_j(M)$ for any other message M with a probability of success greater than random guessing.

Gilbert, MacWilliams, Sloane [71] present techniques which make it *information-theoretically impossible* for an adversary to forge a valid tag with probability greater than random guessing.

Goldreich, Goldwasser, and Micali [74] show how to construct a family of pseudo-random functions from a cryptographically secure pseudo-random bit sequence generator, such that an adversary can not distinguish between $f_j(M)$ and a randomly chosen string of the same length, even if the adversary is first allowed to examine $f_j(x_i)$ for many x_i's of his choice, and is allowed to even pick M (as long as it is different from every x_i he previously asked about). Knowledge of the index j allows efficient computation of $f_j(x)$ for any x. To the adversary (who doesn't know j), the function f_j is indistinguishable in polynomial time from a truly random function (that is, one picked at random from the set of all functions mapping $\{0,1\}^n$ into itself). Since permutations are invertible, the Goldwasser-Goldreich-Micali construction provides a probabilistic private-key cryptosystem, one that is provably secure even against an adaptive chosen ciphertext attack. To send a message M, the sender picks an r at random from the domain of the previously agreed-upon secret pseudo-random function f_j, and then sends the pair $(r, M \oplus f_j(r))$.

Luby and Rackoff [112] have extended the previous result by showing how to construct a family of pseudo-random *permutations* which is secure in the same sense and under the same assumptions. Curiously, their construction is based on the structure of DES.

8 Digital Signatures

The notion of a *digital signature* may prove to be one of the most fundamental and useful inventions of modern cryptography. A signature scheme provides a way for each user to *sign* messages so that the signatures can later be *verified* by anyone else. More specifically, each user can create a matched pair of private and public keys so that only he can create a signature for a message (using his private key), but anyone can verify the signature for the message (using the signer's public key). The verifier can convince himself that the message contents have not been altered since the message was signed. Also, the signer can not later repudiate having signed the message, since no one but the signer possesses his private key.

Diffie and Hellman [53] propose that with a public-key cryptosystem, a user A can sign any message M by appending his digital signature $D_A(M)$ to M. Anyone can check the validity of this signature using A's public key E_A from the public directory, since $E_A(D_A(M)) = M$. Note also that this signature becomes invalid if the message is changed, so that A is protected against modifications after he has signed the message, and the person examining the signature can be sure that the message he has received that was originally signed by A.

By analogy with the paper world, where one might sign a letter and seal it in an envelope, one can sign an electronic message using one's private key, and then *seal* the result by encrypting it with the recipient's public key. The recipient can perform the inverse operations of opening the letter and verifying the signature using his private key and the sender's public key, respectively. These applications of public-key technology to electronic mail are likely to be widespread in the near future.

The RSA public-key cryptosystem allows one to implement digital signatures in a straightforward manner. The private exponent d now becomes the *signing exponent*, and the signature of a message M is now the quantity $M^d \bmod n$. Anyone can verify that this signature is valid using the corresponding public *verification exponent* e by checking the identity $M = (M^d)^e \pmod{n}$. If this equation holds, then the signature M^d must have been created from M by the possessor of the corresponding signing exponent d. (Actually, it is possible that the reverse happened and that the "message" M was computed from the "signature" M^d using the verification equation and the public exponent e. However, such a message is likely to be unintelligible. In practice, this problem is easily avoided by always signing $f(M)$ instead of M, where f is a standard public one-way function.)

If the directory of public keys is accessed over the network, one needs to protect the users from being sent fraudulent messages purporting to be public keys from the directory. An elegant solution is the use of a *certificate* – a copy of a user's public key digitally signed by the public key directory manager or other trusted party. If user A keeps locally a copy of the public key of the directory manager, he can validate all the signed communications from the public-key directory and avoid being tricked into using fraudulent keys. Moreover, each user can transmit the certificate for his public key with any message he signs, thus removing the need for a central directory and allowing one to verify signed messages with no information other than the directory manager's public key. Needham and Schroeder [120] examine some of the protocol issues involved in such a network organization, and compare it to what might be accomplished using conventional cryptography.

Just as some cryptographic schemes are suited for encryption but not signatures, some proposals have been made for *signature-only* schemes. Some early suggestions were made that were based on the use of one-way functions or conventional cryptography [101, 134]. For example, if f is a one-way function, and Alice has published the two numbers $f(x_0) = y_0$ and $f(x_1) = y_1$, then she can sign the message 0 by releasing x_0 and she can similarly sign the message 1 by releasing the message x_1. Merkle [116] introduced some extensions of this basic idea, involving building a tree of authenticated values whose root is stored in the public key of the signer.

Rabin [130] proposed a method where the signature for a message M was essentially the square root of M, modulo n, the product of two large primes. Since the ability to take square roots is provably equivalent to the ability to factor n, an adversary should not be able to forge any signatures unless he can factor n. This argument assumes that the adversary only has access to the public key containing the modulus n of the signer. In practice, an enemey may break this scheme with an *active* attack by asking the real signer to sign $M = x^2 \bmod n$, where x has been chosen randomly. If the signer agrees and produces a square root y of M, there is half a chance that $\gcd(n, x - y)$ will yield a nontrivial factor of n — the signer has thus betrayed his own secrets! Although Rabin proposed some practical techniques for circumventing this problem, they have the effect of eliminating the constructive reduction of factoring to forgery.

In general, knapsack-type schemes are not well-suited for use as signatures schemes, since the mapping $M \cdot \mathbf{a}$ is not "onto", and no effective remedy has been proposed. A review of signature schemes can be found in [84].

8.1 Proving Security of Signature Schemes

A theoretical treatment of digital signatures security was started by Goldwasser, Micali and Yao in [82] and continued in [84, 15],and recently [119]. We first address what attacks are possible on digital signature schemes.

8.1.1 Attacks Against Digital Signatures

We distinguish three basic kinds of attacks, listed below in the order of increasing severity.

- *Key-Only Attack:* In this attack the adversary knows only the public key of the signer and therefore only has the capability of checking the validity of signatures of messages given to him.

- *Known Signature Attack:* The adversary knows the public key of the signer and has seen message/signature pairs chosen and produced by the legal signer. In reality, this the minimum an adversary can do.

- *Chosen Signature Attack:* The adversary is allowed to ask the signer to sign a number of messages of the adversary's choice. The choice of these messages may depend on previously obtained signatures. For example, one may think of a notary public who signs documents on demand.

For a finer subdivision of the adversary's possible attacks see [84].

8.1.2 What does it mean to successfuly forge a signature?

We distinguish several levels of success for an adversary, listed below in the order of increasing success for the adversary.

- *Existential Forgery:* The adversary succeeds in forging the signature of one message, not necessarily of his choice.

- *Selective Forgery:* The adversary succeeds in forging the signature of some message of his choice.

- *Universal Forgery:* The adversary, although unable to find the secret key of the signer, is able to forge the signature of any message.

- *Total Break :* The adversary can compute the signer's secret key.

Clearly, different levels of security may be required for different applications. Sometimes, it may suffice to show that an adversary who is capable of a known signature attack can not succeed in selective forgery, while for other applications (for example when the signer is a notary-public or a tax-return preparer) it may be required that an adversary capable of a chosen signature attack can not succeed even at existential forgery with non-negligible probability.

8.2 Probabilistic Signature Schemes

Probabilistic techniques have also been applied to the creation of digital signatures. This approach was pioneered by Goldwasser, Micali, and Yao [82], who presented signature schemes based on the difficulty of factoring and on the difficulty of inverting the RSA function for which it is provably hard for the adversary to existentially forge using a known signature attack.

Goldwasser, Micali, and Rivest [84] have strengthened this result by proposing a signature scheme which is not existentially forgeable under a chosen message attack. Their scheme is based on the difficulty of factoring, and more generally on the existence of claw-free trapdoor permutations (that is, pairs f_0, f_1 of trapdoor permutations defined on a common domain for which it is hard to find x, y such that $f_0(x) = f_1(y)$).

We briefly sketch their digital signature scheme. Let (f_0, f_1) and (g_0, g_1) be two pairs of claw-free permutations. Let $b = b_1 ... b_k$ be a binary string, and define ${F_b}^{-1}(y) = {f_{b_k}}^{-1}(...({f_{b_2}}^{-1}(f_{b_1}^{-1}(y)))$, and ${G_b}^{-1}(y) = {g_{b_k}}^{-1}(...({g_{b_2}}^{-1}(g_{b_1}^{-1}(y)))$.

The signer makes public (x, f_0, f_1, g_0, g_1) where x is a randomly chosen element in the domain of f_0, f_1, and keeps secret the trapdoor information enabling him to compute F_b^{-1} and G_b^{-1}. The variable *history* is kept by the signer in memory as well, but need not be private. (For simplicity of exposition we assume a prefix-free message space.)

The signature of the ith message m_i is created as follows. The signer:

1. Picks r_i at random in the domain of g_0, g_1 and sets $history = history \cdot r_i$ where $\cdot$ denotes concatenation.

2. Computes $l_i = F_{history}^{-1}(x)$ and $t_i = G_{m_i}^{-1}(r_i)$.

3. Produces the signature of m_i as the triplet $(history, l_i, t_i)$.

To check the validity of the signature (h, l, t) of message m, anyone with access to the signer's public key can check that:

1. $F_h(l) = x$ where x is in the public file.

2. $G_m(t) = r$ where r is a suffix of h.

If both conditions hold the signature is valid.

The scheme as we describe it, although attractive in theory, is quite inefficient. However, it can be modified to allow more compact signatures, to make no use of memory between signatures other than for the public and secret keys, and even to remove the need of making random choices for every new signature. In particular, Goldreich [73] has made suggestions that make the factoring-based version of this scheme more practical while preserving its security properties.

Bellare and Micali in [15] have shown a digital signature scheme whose security can be based on the existence of any trapdoor permutation (a weaker requirement than claw-freeness).

A major leap forward has been recently made by Naor and Yung [119] who have shown how, starting with any *one-way* permutation, to design a digital signature scheme which is not existentially forgeable by chosen message attack.

9 Two-Party Protocols

In this section we sketch a number of cryptographic protocol problems that have been addressed in the literature; see [47] for additional examples.

9.1 Examples

9.1.1 User Identification (Friend-or-Foe)

Suppose A and B share a secret key K. Later, A is communicating with someone and he wishes to verify that it is B. A simple *challenge-response* protocol to achieve this identification goes as follows:

- A generates a random value r and transmit r to the other party.
- The other party (assuming it is B) encrypts r using the shared secret key K and transmits the resulting ciphertext back to A.
- A compares the received ciphertext with the result he obtains by encrypting r himself using the secret key K. If they agree, knows that the other party is B; otherwise he assumes that the other party is an impostor.

This protocol is generally more useful than the transmission of an unencrypted shared password from B to A, since an eavesdropper could learn the password and then pretend to be B later. With the challenge-response protocol an eavesdropper presumably learns nothing about K by hearing many values of r encrypted with K as key.

9.1.2 Mental Poker

How might one play a game of cards, such as poker, over the telephone? The difficulty, of course, is in dealing the cards. Shamir, Rivest, and Adleman [145] proposed the following simple strategy (here Alice and Bob are the two players):

- The players jointly select 52 distinct messages $M_1, \ldots, M_{52}$ to represent the cards, and a large prime p.
- The players secretly choose exponents e_A and e_B respectively, so that Alice has a secret encryption function $E_A(M) = M^{e_A} \bmod p$ (similarly for Bob). Each player computes the corresponding decryption exponents d_A, d_B defining decryption functions $D_A(C) = C^{d_A} \bmod p$ (and similarly for Bob).

- Alice (the dealer) encrypts the cards $M_1, \ldots, M_{52}$, shuffles them (permutes their order), and sends the resulting list to Bob.

- Bob selects five of the cards and returns them to Alice; she decrypts these for her hand.

- Bob selects five of the remaining cards for his own hand, encrypts them with his encryption function, and sends the result to Alice. Note that each such card has the form $E_B(E_A(M_i)) = E_A(E_B(M_i))$ since E_A and E_B commute.

- Alice decrypts the five cards with D_A, and returns the result to Bob, who decrypts them with D_B to obtain his hand.

At the end of the play, the parties can reveal their secret keys so they can assure themselves that the other hasn't cheated.

This technique requires the use of *commutative* encryption functions. The particular scheme as proposed may be less satisfactory than desired (depending on the coding of the cards) since, as pointed out by Lipton [108], M will have Jacobi symbol 1 if and only if $E_A(M)$ does — the function E_A *leaks a bit*. Another attack has been proposed by Coppersmith [42]. A number of authors have proposed extensions and variations of this protocol.

9.1.3 Coin Flipping

Blum [24] has proposed the problem of *coin flipping over the telephone*. If Alice and Bob don't trust each other, then if they wish to flip a coin they need a procedure that will produce an outcome (head or tails) that can not be biased by either party.

Using probabilistic encryption, Alice could send encrypted versions of the messages "Heads" and "Tails" to Bob. Bob picks one of the ciphertexts, and indicates his choice to Alice. Alice then reveals the secret encryption key to Bob. There are a number of interesting variations and subtleties to this problem (see, for example, [79, 40]).

9.1.4 Oblivious Transfer

An *oblivious transfer* is an unusual protocol wherein Bob transfers a message M to Alice in such a way that

- With probability 1/2, Alice receives the message, and with probability 1/2, Alice receives garbage instead.

- At the end of protocol Bob doesn't know whether or not Alice received the message.

This strange-sounding protocol has a number of useful applications (see, for example [131, 21]). In fact, Kilian has recently shown [97] that the ability to perform oblivious transfers is a sufficiently strong primitive to enable *any* two-party protocol to be performed.

9.1.5 Other examples

The problem of *contract signing* is that of simultaneously exchanging digital signatures to a contract. That is, we assume that an ordinary digital signature scheme is available, and want to arrange a two-party protocol so that the signatures are effectively simultaneous — neither party obtains the other's signature before giving up his own. Several interesting solutions to the problem have been proposed (see [62] or [19]).

The *certified electronic mail problem* is similar to the contract-signing problem above. The goal is to achieve a simultaneous exchange of an electronic letter M and a signed receipt for M from the recipient.

Another exchange problem is the simultaneous exchange of secrets. This has been studied in [25, 155, 110, 161].

9.2 Zero-Knowledge Protocols

The previous section listed a number of cryptographic protocol applications. In this section we review the theory that has been developed to prove that these protocols are secure, and to design protocols that are "provably secure by construction".

9.2.1 Zero-Knowledge Interactive Proofs

An elegant way to prove a cryptographic protocol secure is to prove that it is a *zero-knowledge protocol.* Informally, a protocol is a zero-knowledge protocol if one party learns nothing (zero) from the protocol above and beyond what he is supposed to learn. This theory was originated by Goldwasser, Micali, and Rackoff [80], and has been extended by many others, including Galil, Haber, and Yung [67], Chaum [34], Goldwasser and Sipser [83], and Brassard and Crepeau [31].

Zero-knowledge protocols are two-party protocols; one party is called the *prover* and the other the *verifier.* The prover knows some fact and wishes to convince the verifier of this fact. The verifier wants a protocol that will allow the prover to convince him of the validity of the fact, if and only if the fact is true. More precisely, the prover (if he follows the protocol) will be able (with extremely high probability) to convince the verifier of the validity of the fact if the fact is true, but the prover (even if he attempts to cheat) will not have any significant chance of convincing the verifier of the validity of the fact if the fact is false. However, the prover doesn't wish to disclose any information above and beyond the validity of the fact itself (for example, the reason why the fact holds). This condition can be very useful in cryptographic protocols, as we shall see.

Examples of nontrivial NP languages for which there exist zero-knowledge protocols include quadratic residuosity and graph-isomorphism. (Curiously, the complements of both of these languages also possess zero-knowledge proofs.)

More generally, it has been shown by Goldreich, Micali, and Wigderson [76] that *every* language in NP possesses a zero-knowledge protocol, on the assumption that there secure encryption is possible. (Probabilistic encryption schemes work fine here.)

As a concrete example of a zero-knoledge protocol, suppose I wish to convince you that a certain input graph is three-colorable, without revealing to you the coloring that I know. I can do so in a sequence of $|E|^2$ stages, each of which goes as follows.

- I switch the three colors at random (e.g., switching all red nodes to blue, all blue nodes to yellow, and all yellow nodes to red).
- I encrypt the color of each node, using a different probabilistic encryption scheme for each node, and show you all these encryptions, together with the correspondence indicating which ciphertext goes with which vertex.
- You select an edge of the graph.
- I reveal the decryptions of the colors of the two nodes that are incident to this edge by revealing the corresponding decryption keys.
- You confirm that the decryptions are proper, and that the two endpoints of the edge are colored with two different but legal colors.

If the graph is indeed three-colorable (and I know the coloring), then you will never detect any edge being incorrectly labelled. However, if the graph is not three-colorable, then there is a chance of at least $|E|^{-1}$ on each stage that I will be caught trying to fool you. The chance that I could fool you for $|E|^2$ stages without being caught is exponentially small.

Note that the history of our communications—in the case that the graph is three-colorable—consists of the concatenation of the messages sent during each stage. It is possible to prove (on the assumption that secure encryption is possible) that the probability distribution defined over these histories by our set of possible interactions is indistinguishable in polynomial time from a distribution that you can create on these histories by yourself, without my participation. This fact means that you gain zero (additional) knowledge from the protocol, other than the fact that the graph is three-colorable.

The proof that graph three-colorability has such a zero-knowledge interactive proof system can be used to prove that every language in NP has such a zero-knowledge proof system.

9.2.2 Applications to User Identification

Zero knowledge proofs provide a revolutionary new way to realize passwords [80, 64]. The idea is for every user to store a statement of a theorem in his publicly readable directory, the proof of which only he knows. Upon login, the user engages in a zero-knowledge proof of the correctness of the theorem. If the proof is convincing, access permission is granted. This guarantees that even an adversary who overhears the zero-knowledge proof can not learn enough to gain unauthorized access. This is a novel property which can not be achieved with traditional password mechanisms. Recently, Fiat and Shamir [64] have developed variations on some of the previously proposed zero-knowledge protocols [80] which are quite efficient and particularly useful for user identification and passwords.

10 Multiparty Protocols

In a typical multi-party protocol problem, a number of parties wish to coordinate their activities to achieve some goal, even though some (sufficiently small) subset of them may have been corrupted by an adversary. The protocol should guarantee that the "good" parties are able to achieve the goal even though the corrupted parties send misleading information or otherwise maliciously misbehave in an attempt prevent the good parties from succeeding.

10.1 Examples

10.1.1 Secret Sharing

In 1979 Shamir [141] considered the problem of *sharing a secret*, defined as follows. Suppose n people wish to share a secret (such as a secret cryptographic key) by dividing it into pieces in such a way that *any* subset of k people can recreate the secret from their pieces, but *no* subset of less than k people can do so. Here k is a given fixed positive integer less than n.

Shamir proposed the following elegant solution to this problem. Let the secret be represented as an integer s, which is less than some large convenient prime p. The secret can be "divided into pieces" by

1. Generating k coefficients $a_0, a_1, \ldots, a_{k-1}$, where $a_0 = s$ but all the other coefficients are randomly chosen modulo p.
2. Defining the polynomial $f(x)$ to be $\sum_{0 \leq i < k} a_i x^i \bmod p$.
3. Giving party i the "piece" $f(i)$, for $1 \leq i \leq n$.

Now, given any k values for f, one can interpolate to find f's coefficients (including the secret $a_0 = s$). However, a subset of $k-1$ values for f provides absolutely no information about s, since for any possible s there is a polynomial of degree $k-1$ consistent with the given values and the possible value for s.

Shamir's scheme suffers from two problems. If the dealer of the secret is dishonest, he can give pieces which when put together do not uniquely define a secret. Secondly, if some of the players are dishonest, at the reconstruction stage they may provide other players with different pieces than they received and again cause an incorrect secret to be reconstructed.

Chor, Goldwasser, Micali, and Awerbuch [38] have observed the above problems and showed how to achieve secret sharing based on the intractability of factoring which does not suffer from the above problems. They call the new protocol *verifiable secret sharing* since now every party can verify that the piece of the secret he received is indeed a proper piece. Their protocol tolerated up to $O(\log n)$ colluders. Benaloh [16], and others [76, 63] showed how to achieve verifiable secret sharing if any one-way function exists which tolerates a minority of colluders. In [20] it has been recently shown how to achieve verifiable secret sharing against a minority of colluders using error correcting codes, without making cryptographic assumptions.

10.1.2 Anonymous Transactions

Chaum has advocated the use of *anonymous transactions* as a way of protecting individuals from the maintenance by "Big Brother" of a database listing all their transactions, and proposes using *digital pseudonyms* to do so. Using pseudonyms, individuals can enter into electronic transactions with assurance that the transactions can not be later traced to the individual. However, since the individual is anonymous, the other party may wish assurance that the individual is authorized to enter into the transaction, or is able to pay. [35, 33].

10.1.3 Voting

Cryptographic technology can be used to manage an election so that every voter's vote remains private, but yet every voter can be sure that the vote-counting was not manipulated. (See Cohen and Fischer [41].)

10.2 Multiparty Ping-Pong Protocols

One way of demonstrating that a cryptographic protocol is secure is to show that the primitive operations that each party performs can not be composed to reveal any secret information.

Consider a simple example due to Dolev and Yao [57] involving the use of public keys. Alice sends a message M to Bob, encrypting it with his public key, so that the ciphertext C is $E_B(M)$ where E_B is Bob's public encryption key. Then Bob "echos" the message back to Alice, encrypting it with Alice's public key, so that the ciphertext returned is $C' = E_A(M)$. This completes the description of the protocol.

Is this secure? Since the message M is encrypted on both trips, it is clearly infeasible for a *passive* eavesdropper to learn M. However, an *active* eavesdropper X can defeat this protocol. Here's how: the eavedropper X overhears the previous conversation, and records the ciphertext $C = E_B(M)$. Later, X starts up a conversation with Bob using this protocol, and sends Bob the encrypted message $E_B(M)$ that he has recorded. Now Bob dutifully returns to X the ciphertext $E_X(M)$, which gives X the message M he desires!

The moral is that an adversary may be able to "cut and paste" various pieces of the protocol together to break the system, where each "piece" is an elementary transaction performed by a legitimate party during the protocol, or a step that the adversary can perform himself.

It is sometimes possible to *prove* that a protocol is invulnerable to this style of attack. Dolev and Yao [57] pioneered this style of proof; additional work was performed by Dolev, Even, and Karp [56], Yao [162], and Even and Goldreich [61]. In other cases a modification of the protocol can eliminate or alleviate the danger; see [136] as an example of this approach against the danger of an adversary "inserting himself into the middle" of a public-key exchange protocol.

10.3 Multiparty Protocols When Most Parties are Honest

Goldreich, Micali, and Wigderson [76] have shown how to "compile" a protocol designed for honest parties into one which will still work correctly even if some number less than half of the players try to "cheat". While the protocol for the honest parties may involve the disclosure of secrets, at the end of the compiled protocol none of the parties know any more than what they knew originally, plus whatever information is disclosed as the "official output" of the protocol. Their compiler correctness and privacy is based on the existence of trapdoor functions.

Ben-Or, Goldwasser and Wigderson [20] and Chaum, Creapau, and Damgård [36] go one step further. They assume secret communication between pairs of users as a primitive. Making no intractability assumption, they show a "compiler" which, given a description (e.g., a polynomial time algorithm or circuit) of any polynomial time function f, produces a protocol which always computes the function correctly and gurantees that no additional information to the function value is leaked to dishonest players . The "compiler" withstands up to 1/3 of the parties acting dishonestly in a manner directed by a worst-case unbounded-computation-time adverasry.

These "master theorems" promise to be very powerful tool in the future design of secure protocols.

11 Cryptography and Complexity Theory

Some cryptographic operators are secure in an information-theoretic sense; examples are the Vernam one-time pad, the secret-sharing scheme of Shamir, and the authentication scheme of Gilbert et al.. In these cases the adversary never obtains enough information to enable him to set up a problem having a unique solution. As a consequence no amount of computational power can help him resolve this intrinsic uncertainty.

However, most practical cryptographic schemes are not information-theoretically secure. While the adversary is given enough information to determine a unique solution, the problem of actually computing this solution is deemed to be computationally intractable. We may term this *computational security.* Computational security depends critically on the existence and careful exploitation of "hard" computational problems.

While one goal of computational complexity theory is to be able to precisely characterize the computational difficulty of arbitrary problems, the state of this theory is such that we can currently only derive some tools and suggestive guidelines.

For example, we may observe that if it turns out to be the case that $P = NP$ (see [68]), then computational security would be unachievable, since the adversary's problem is easily seen to be in NP. (The adversary can just guess the secret keys, check their correctness against some known plaintext/ciphertext pairs, and thereby "break" the system.)

If we assume that $P \neq NP$, then it would seem natural to try to design cryptographic systems such that the problem of breaking them is NP-complete. However, this is difficult to arrange, since cryptographic problems usually have *unique* solutions, whereas NP-complete problems may have many solutions. It is not easy to reduce a problem with many solutions to a problem having a unique solution. Grollman and Selman [87] show that one-way functions exist if and only if $P \neq UP$, where UP is the class of languages accepted by a nondeterministic Turing machine which has at most one accepting computation for any input. They also show that secure public-key cryptosystems exist only if $P \neq UP$. Lempel and Yacobi [102] present a curious cryptographic system for which the cryptanalytic problem is NP-complete in general, even for chosen-plaintext attacks, but which is likely to be easily breakable in practice.

The Lempel/Yacobi example illustrates another difficulty with attempting to use the traditional notions of computational complexity in the design of cryptographic systems: the traditional notions relate to the *worst-case* complexity of the problem, whereas cryptographic security more realistically depends on the *average-case* complexity of the problem. There is much yet to be learned about the average-case complexity of problems. However, Levin [107] has introduced a formal notion of what it means for a problem to be "complete" in the sense of its average-case complexity.

Brassard [30] has shown that in certain relativized models of computation secure cryptography is possible.

Yao [163] develops the theory of trapdoor functions and pseudorandom bit sequence generators, and shows that if a strong one-way function exists, then

$$R \subseteq \bigcap_{\epsilon > 0} \mathrm{DTIME}(2^{n^{\epsilon}}), \tag{9}$$

since any algorithm in R can be adapted to use a pseudo-random bit sequence generator with a seed of size n^{ϵ} instead of a true random bit generator. (Note that all seeds have to be tried.)

Acknowledgements

I would like to thank Shafi Goldwasser for her extensive help in preparing this paper, and also Benny Chor, Oded Goldreich, Silvio Micali, Phil Rogaway, and Alan Sherman for their comments and suggestions for improvements.

References

[1] L. M. Adleman. On breaking generalized knapsack public key cryptosystems. In *Proc. 15th ACM Symposium on Theory of Computing*, pages 402–412, ACM, Boston, 1983.

[2] L. M. Adleman. A subexponential algorithm for the discrete logarithm problem with applications to cryptography. In *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science*, pages 55–60, IEEE, Providence, 1977.

[3] L. M. Adleman and M. A. Huang. Recognizing primes in random polynomial time. In *Proc. 19th ACM Symposium on Theory of Computing*, pages 462–469, ACM, New York City, 1987.

[4] L. M. Adleman, K. Manders, and G. Miller. On taking roots in finite fields. In *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science*, pages 175–177, IEEE, Providence, 1977.

[5] L. M. Adleman, C. Pomerance, and R. S. Rumely. On distinguishing prime numbers from composite numbers. *Ann. Math.*, 117:173–206, 1983.

[6] W. B. Alexi, B. Chor, O. Goldreich, and C. P. Schnorr. RSA and Rabin functions: certain parts are as hard as the whole. *SIAM J. Computing*, 17(2):194–209, April 1988.

[7] B. Alpern and F. B. Schneider. Key exchange using 'keyless cryptography'. *Information Processing Letters*, 16:79–81, 1983.

[8] D. Angluin. *Lecture notes on the complexity of some problems in number theory.* Technical Report TR-243, Yale University Computer Science Department, August 1982.

[9] D. Angluin and D. Lichtenstein. *Provable Security of Cryptosystems: A Survey.* Technical Report TR-288, Yale University Department of Computer Science, October 1983.

[10] C. A. Asmuth and G. R. Blakley. An efficient algorithm for constructing a cryptosystem which is harder to break than two other cryptosystems. *Comp. and Maths. with Appls.*, 7:447–450, 1981.

[11] Eric Bach. How to generate factored random numbers. *SIAM J. Computing*, 17(2):179–193, April 1988.

[12] James Bamford. *The Puzzle Palace – A Report on NSA, America's Most Secret Agency.* Houghton Miflin, Boston, 1982.

[13] H. Beker and F. Piper. *Cipher Systems: The Protection of Communications.* Northwood, London, 1982.

[14] D. A. Bell and S. E. Olding. *An Annotated Bibliography of Cryptography.* Technical Report COM-100, National Physical Laboratory, January 1978.

[15] Mihir Bellare and Silvio Micali. How to sign given any trapdoor function. In *Proc. 20th ACM Symposium on Theory of Computing*, pages 32–42, ACM, Chicago, 1988.

[16] J. Benaloh. Secret sharing homomorphisms: keeping shares of a secret sharing. In A. M. Odlyzko, editor, *Proceedings CRYPTO 86*, Springer, 1987. Lecture Notes in Computer Science No. 263.

[17] C. H. Bennett, G. Brassard, S. Breidbard, and S. Wiesner. Quantum cryptography. In *Proceedings CRYPTO 82*, pages 267–275, Plenum Press, 1983.

[18] M. Ben-Or, B. Chor, and A. Shamir. On the cryptographic security of single RSA bits. In *Proc. 15th ACM Symposium on Theory of Computing*, pages 421–430, ACM, Boston, 1983.

[19] M. Ben-Or, O. Goldreich, S. Micali, and R. L. Rivest. A fair protocol for signing contracts. In *ICALP*, pages 43–52, 1985.

[20] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for fault-tolerant distributed computing. In *Proc. 20th ACM Symposium on Theory of Computing*, pages 1–10, ACM, Chicago, 1988.

[21] R. Berger, R. Peralta, and T. Tedrick. A provably secure oblivious transfer protocol. In T. Beth, N. Cot, and I. Ingemarsson, editors, *Proceedings of EUROCRYPT 84*, pages 379–386, Springer, 1985.

[22] E. Berlekamp. Factoring polynomials over large finite fields. *Math. Comp*, 24:713–735, 1970.

[23] L. Blum, M. Blum, and M. Shub. A simple unpredictable pseudo-random number generator. *SIAM J. Computing*, 15(2):364–383, May 1986.

[24] M. Blum. Coin flipping by telephone. In *Proc. IEEE Spring COMPCOM*, pages 133–137, IEEE, 1982.

[25] M. Blum. How to exchange (secret) keys. *Trans. Computer Systems*, 1:175–193, May 1983.

[26] M. Blum. Independent unbiased coin flips from a correlated biased source: a finite state Markov chain. In *Proceedings of the* 25*th IEEE Symposium on Foundations of Computer Science*, pages 425–433, IEEE, Singer Island, 1984.

[27] M. Blum and S. Goldwasser. An efficient probabilistic public-key encryption scheme which hides all partial information. In G. R. Blakley and D. C. Chaum, editors, *Proceedings CRYPTO 84*, pages 289–302, Springer, 1985. Lecture Notes in Computer Science No. 196.

[28] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM J. Computing*, 13(4):850–863, November 1984.

[29] G. Brassard. *Introduction to Modern Cryptology.* Technical Report 606, Départment d'informatique et de recherche opérationelle, 1987.

[30] G. Brassard. Relativized cryptography. In *Proc.* 20*th Annual IEEE Symposium on Foundations of Computer Science*, pages 383–391, IEEE, San Juan, Puerto Rico, 1979.

[31] G. Brassard and C. Crepeau. Nontransitive transfer of confidence: a perfect zero-knowledge interactive protocol for SAT and beyond. In *Proceedings of the* 27*th IEEE Symposium on Foundations of Computer Science*, pages 188–195, IEEE, Toronto, 1986.

[32] E. F. Brickell. Breaking iterated knapsacks. In G. R. Blakley and D. C. Chaum, editors, *Proceedings CRYPTO 84*, pages 342–358, Springer, 1985. Lecture Notes in Computer Science No. 196.

[33] D. Chaum. Blind signatures for untraceable payments. In R. L. Rivest, A. Sherman, and D. Chaum, editors, *Proceedings CRYPTO 82*, pages 199–204, Plenum Press, New York, 1983.

[34] D. Chaum. Demonstrating that a public predicate can be satisfied without revealing any information about how. In A. M. Odlyzko, editor, *Proceedings CRYPTO 86*, pages 195–199, Springer, 1987. Lecture Notes in Computer Science No. 263.

[35] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communcations of the ACM*, 24:84–88, February 1981.

[36] D. Chaum, C. Crepeau, and I. Damgård. Multi-party unconditionally secure protocols. In *Proc.* 20*th ACM Symposium on Theory of Computing*, ACM, Chicago, 1988.

[37] B. Chor and O. Goldreich. Unbiased bits from sources of weak randomness and probabilistic communication complexity. *SIAM J. Computing*, 17(2):230–261, April 1988.

[38] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *Proceedings of the* 26*th IEEE Symposium on Foundations of Computer Science*, pages 383–395, IEEE, Portland, 1985.

[39] B. Chor and R. L. Rivest. A knapsack type public-key cryptosystem based on arithmetic in finite fields. *IEEE Trans. Inform. Theory*, 34(5):901–909, September 1988.

[40] R. Cleve. Limits on the security of coin flips when half the processors are faulty. In *Proc.* 18*th ACM Symposium on Theory of Computing*, pages 364–369, ACM, Berkeley, 1986.

[41] J. D. Cohen and M. J. Fischer. A robust and verifiable cryptographically secure election scheme. In *Proceedings of the* 26*th IEEE Symposium on Foundations of Computer Science*, pages 372–382, IEEE, Portland, 1985.

[42] D. Coppersmith. Cheating at mental poker. In A. M. Odlyzko, editor, *Proceedings CRYPTO 86*, pages 104–107, Springer, 1987. Lecture Notes in Computer Science No. 263.

[43] D. Coppersmith. Evaluating logarithms in $GF(2^n)$. In *Proc.* 16*th ACM Symposium on Theory of Computing*, pages 201–207, ACM, Washington, D.C., 1984.

[44] D. Coppersmith, A. M. Odlyzko, and R. Schroeppel. Dicrete logarithms in $GF(p)$. *Algorithmica*, 1(1):1–16, 1986.

[45] D. W. Davies, editor. *Tutorial: The Security of Data in Networks.* IEEE, 1981. IEEE Computer Society Order #366.

[46] R. A. DeMillo, D. P. Dobkin, A. Jones, and R. J. Lipton, editors. *Foundations of Secure Computation.* Academic Press, New York, 1978.

[47] R. A. DeMillo, N. Lynch, and M. J. Merritt. Cryptographic protocols. In *Proc.* 14*th ACM Symposium on Theory of Computing*, pages 383–400, ACM, San Francisco, 1982.

[48] D. E. Denning. *Cryptography and Data Security.* Addison-Wesley, Reading, Mass., 1982.

[49] D. E. Denning and P. J. Denning. Data security. *ACM Computing Surveys*, 11:227–249, September 1979.

[50] W. Diffie and M. E. Hellman. Exhaustive cryptanalysis of the NBS data encryption standard. *Computer*, 10:74–84, June 1977.

[51] W. Diffie and M. E. Hellman. An introduction to cryptography. In Slonim, Unger, and Fisher, editors, *Advances in Data Communication Management*, chapter 4, pages 44–134, Wiley, 1984.

[52] W. Diffie and M. E. Hellman. Multiuser cryptographic techniques. In *Proceedings AFIPS 1976 National Computer Conference*, pages 109–112, AFIPS, Montvale, N.J., 1976.

[53] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Trans. Inform. Theory*, IT-22:644–654, November 1976.

[54] W. Diffie and M. E. Hellman. Privacy and authentication: an introduction to cryptography. *Proceedings of the IEEE*, 67:397–427, March 1979.

[55] John D. Dixon. Factorization and primality tests. *The American Mathematical Monthly*, 91(6):333–352, June-July 1984.

[56] D. Dolev, S. Even, and R. M. Karp. On the security of ping-pong protocols. In R. L. Rivest, A. Sherman, and D. Chaum, editors, *Proceedings CRYPTO 82*, pages 177–186, Plenum Press, New York, 1983.

[57] D. Dolev and A. C. Yao. On the security of public key protocols. In *Proceedings of the* 22*nd IEEE Symposium on Foundations of Computer Science*, pages 350–357, IEEE, Nashville, 1981.

[58] P. Elias. The efficient construction of an unbiased random sequence. *Ann. Math. Statist.*, 43(3):865–870, 1972.

[59] A. Evans, W. Kantrowitz, and E. Weiss. A user authentication scheme not requiring secrecy in the computer. *CACM*, 17:437–442, August 1974.

[60] S. Even and O. Goldreich. On the power of cascade ciphers. *ACM Trans. Computer Systems*, 3:108–116, May 1985.

[61] S. Even and O. Goldreich. On the security of multi-party ping-pong protocols. In *Proceedings of the* 24*th IEEE Symposium on Foundations of Computer Science*, pages 34–39, IEEE, Tucson, 1983.

[62] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. In R. L. Rivest, A. Sherman, and D. Chaum, editors, *Proceedings CRYPTO 82*, pages 205–210, Plenum Press, New York, 1983.

[63] P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *Proceedings of the* 28*th IEEE Symposium on Foundations of Computer Science*, pages 427–438, IEEE, Los Angeles, 1985.

[64] A. Fiat and A. Shamir. How to prove yourself: practical solutions to identification and signature problems. In A. M. Odlyzko, editor, *Proceedings CRYPTO 86*, pages 186–194, Springer, 1987. Lecture Notes in Computer Science No. 263.

[65] A. M. Frieze, J. Hastad, R. Kannan, J. C. Lagarias, and A. Shamir. Reconstructing truncated integer variables satisfying linear congruences. *SIAM J. Computing*, 17(2):262–280, April 1988.

[66] H. F. Gaines. *Cryptanalysis: A Study of Ciphers and Their Solutions.* Dover, 1956.

[67] Z. Galil, S. Haber, and M. Yung. A private interactive test of a boolean predicate and minimum-knowledge public-key cryptosystems. In *Proceedings of the* 26*th IEEE Symposium on Foundations of Computer Science*, pages 360–371, IEEE, Portland, 1985.

[68] M. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, 1979.

[69] J. Garliński. *Intercept: The Enigma War.* Dent, London, 1979.

[70] L. A. Gerhardt and R. C. Dixon, editors. *Spread Spectrum Communications.* Volume COM25, IEEE, August 1977. Special issue of IEEE Trans. Comm.

[71] E. N. Gilbert, F. J. MacWilliams, and N. J. A. Sloane. Codes which detect deception. *Bell System Tech. J.*, 53:405–424, 1974.

[72] J. Gill. Computational complexity of probabilistic Turing machines. *SIAM J. Computing*, 6:675–695, December 1977.

[73] O. Goldreich. *Two remarks concerning the Goldwasser-Micali-Rivest signature scheme.* Technical Report MIT/LCS/TM-315, MIT Laboratory for Computer Science, September 1986.

[74] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. In *Proceedings of the* 25*th IEEE Symposium on Foundations of Computer Science*, pages 464–479, IEEE, Singer Island, 1984.

[75] O. Goldreich and L. Levin. A hard-core predicate for all one-way functions. 1989. To appear in STOC '89.

[76] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design. In *Proceedings of the* 27*th IEEE Symposium on Foundations of Computer Science*, pages 174–187, IEEE, Toronto, 1986.

[77] S. Goldwasser and J. Kilian. Almost all primes can be quickly certified. In *Proc.* 18*th ACM Symposium on Theory of Computing*, pages 316–329, ACM, Berkeley, 1986.

[78] S. Goldwasser and S. Micali. Probabilistic encryption. *JCSS*, 28(2):270–299, April 1984.

[79] S. Goldwasser and S. Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *Proc.* 14*th ACM Symposium on Theory of Computing*, pages 365–377, ACM, San Francisco, 1982.

[80] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. *SIAM. J. Computing*, 18(1):186–208, February 1989.

[81] S. Goldwasser, S. Micali, and P. Tong. Why and how to establish a private code on a public network. In *Proceedings of the* 23*rd IEEE Symposium on Foundations of Computer Science*, pages 134–144, IEEE, Chicago, 1982.

[82] S. Goldwasser, S. Micali, and A. Yao. Strong signature schemes. In *Proc.* 15*th ACM Symposium on Theory of Computing*, pages 431–439, ACM, Boston, 1983.

[83] S. Goldwasser and M. Sipser. Private coins versus public coins in interactive proof systems. In *Proc.* 18*th ACM Symposium on Theory of Computing*, pages 59–68, ACM, Berkeley, 1986.

[84] Shafi Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Computing*, 17(2):281–308, April 1988.

[85] S. W. Golomb. *Shift Register Sequences.* Aegean Park Press, Laguna Hills, 1982. Revised edition.

[86] G. Goos and J. Hartmanis, editors. *Cryptography: Proceedings of the Workshop on Cryptography.* Springer, Burg Feuerstein, 1983. Lecture Notes in Computer Science No. 149.

[87] J. Grollman and A. L. Selman. Complexity measures for public-key cryptosystems. *SIAM J. Computing*, 17(2):309–335, April 1988.

[88] J. Hastad. Solving simultaneous modular equations of low degree. *SIAM J. Computing*, 17(2):336–341, April 1988.

[89] M. E. Hellman. A cryptanalytic time-memory trade off. *IEEE Trans. Inform. Theory*, IT-26:401–406, 1980.

[90] M. E. Hellman. An extension of the Shannon theory approach to cryptography. *IEEE Trans. Inform. Theory*, IT-23:289–294, 1977.

[91] M. E. Hellman. The mathematics of public key cryptography. *Scientific American*, 241:146–157, February 1979.

[92] D. Kahn. *The Codebreakers.* Macmillian, New York, 1967.

[93] B. S. Kaliski, Jr. *Elliptic Curves and Cryptography: A Pseudorandom Bit Generator and Other Tools.* PhD thesis, MIT EECS Dept., January 1988. Published as MIT LCS Technical Report MIT/LCS/TR-411 (Jan. 1988).

[94] B. S. Kaliski, Jr. A pseudo-random bit generator based on elliptic logarithms. In A. M. Odlyzko, editor, *Proceedings CRYPTO 86*, pages 84–103, Springer, 1987. Lecture Notes in Computer Science No. 263.

[95] B. S. Kaliski, Jr., R. L. Rivest, and A. Sherman. Is DES a pure cipher? (results of more cycling experiments on DES),. In H. C. Williams, editor, *Proceedings CRYPTO 85*, pages 212–226, Springer, 1986. Lecture Notes in Computer Science No. 218.

[96] R. Kannan, A. Lenstra, and L. Lovász. Polynomial factorization and non-randomness of bits of algebraic and some transcendental numbers. In *Proc. 16th ACM Symposium on Theory of Computing*, pages 191–200, ACM, Washington, D.C., 1984.

[97] J. Kilian. Founding cryptography on oblivious transfer. In *Proc. 20th ACM Symposium on Theory of Computing*, pages 20–31, ACM, Chicago, 1988.

[98] D. E. Knuth. *The Art of Computer Programming: Vol. 2, Seminumerical Algorithms.* Addison-Wesley, 1969.

[99] A. G. Konheim. *Cryptography: A Primer.* Wiley, 1981.

[100] J. C. Lagarias and A. M. Odlyzko. Solving low-density subset sum problems. In *Proceedings of the 24th IEEE Symposium on Foundations of Computer Science*, pages 1–10, IEEE, Tucson, 1983.

[101] L. Lamport. *Constructing Digital Signatures from a One-Way Function.* Technical Report CSL-98, SRI International, October 1979.

[102] A. Lempel. Cryptology in transition: a survey. *Computing Surveys*, 11:285–304, December 1979.

[103] A. K. Lenstra and H. W. Lenstra, Jr. Algorithms in number theory. In *this volume*, North-Holland, 1989.

[104] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Methematische Ann.*, 261:513–534, 1982.

[105] W. J. LeVeque. *Fundamentals of Number Theory.* Addison-Wesley, 1977.

[106] L. A. Levin. One-way functions and pseudorandom generators. In *Proc. 17th ACM Symposium on Theory of Computing*, pages 363–365, ACM, Providence, 1985.

[107] L. A. Levin. Problems, complete in 'average' instance. In *Proc. 16th ACM Symposium on Theory of Computing*, page 465, ACM, Washington, D.C., 1984.

[108] R. Lipton. How to cheat at mental poker. In *Proc. AMS Short Course on Cryptography*, 1981.

[109] D. L. Long and A. Wigderson. The discrete logarithm problem hides $O(\log n)$ bits. *SIAM J. Computing*, 17(2):363–372, April 1988.

[110] M. Luby, S. Micali, and C. Rackoff. How to simultaneously exchange a secret bit by flipping a symmetrically biased coin. In *Proceedings of the 24th IEEE Symposium on Foundations of Computer Science*, pages 11–22, IEEE, Tucson, 1983.

[111] M. Luby and C. Rackoff. How to construct pseudorandom permutations and pseudorandom functions. *SIAM J. Computing*, 17(2):373–386, April 1988.

[112] M. Luby and C. Rackoff. Pseudo-random permutation generators and cryptographic composition. In *Proc. 18th ACM Symposium on Theory of Computing*, pages 356–363, ACM, Berkeley, 1986.

[113] R. J. McEliece. *A Public-Key System Based on Algebraic Coding Theory*, pages 114–116. Jet Propulsion Lab, 1978. DSN Progress Report 44.

[114] R. Merkle and M. Hellman. Hiding information and signatures in trapdoor knapsacks. *IEEE Trans. Inform. Theory*, IT-24:525–530, September 1978.

[115] R. C. Merkle. Secure communications over insecure channels. *Communcations of the ACM*, 21:294–299, April 1978.

[116] Ralph Charles Merkle. *Secrecy, Authentication, and Public Key Systems.* Technical Report, Stanford University, June 1979.

[117] C. H. Meyer and S. M. Matyas. *Cryptography: A New Dimension in Computer Data Security.* John Wiley and Sons, New York, 1982.

[118] S. Micali, C. Rackoff, and R. H. Sloan. The notion of security for probabilistic cryptosystems. *SIAM J. Computing*, 17(2):412–426, April 1988.

[119] M. Naor and M. Yung. Universal one-way hash functions and their cryptographic applications. In *Proc. 21th ACM Symposium on Theory of Computing*, ACM, Seattle, 1989.

[120] R. M. Needham and M. D. Schroeder. Using encryption for authentication in large networks of computers. *Communcations of the ACM*, 21(12):993–999, December 1978.

[121] I. Niven and H. S. Zuckerman. *An Introduction to the Theory of Numbers.* Wiley, 1972.

[122] A. M. Odlyzko. Cryptanalytic attacks on the multiplicative knapsack scheme and on Shamir's fast signature scheme. *IEEE Trans. Inform. Theory*, IT-30:594–601, July 1984.

[123] A. M. Odlyzko. Discrete logarithms in finite fields and their cryptographic significance. In T. Beth, N. Cot, and I. Ingemarsson, editors, *Proceedings of EUROCRYPT 84*, pages 224–314, Springer, Paris, 1985. Lecture Notes in Computer Science No. 209.

[124] National Bureau of Standards. *Announcing the Data Encryption Standard*. Technical Report FIPS Publication 46, National Bureau of Standards, January 1977.

[125] J. Plumstead. Inferring a sequence generated by a linear congruence. In *Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science*, pages 153–159, IEEE, Chicago, 1982.

[126] S. C. Pohlig and M. E. Hellman. An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance. *IEEE Trans. Inform. Theory*, IT-24:106–110, January 1978.

[127] C. Pomerance. Analysis and comparison of some integer factoring algorithms. In H. W. Lenstra, Jr. and R. Tijdeman, editors, *Computational Methods in Number Theory*, pages 89–139, Math. Centrum Tract, Amsterdam, 1982.

[128] C. Pomerance, J. W. Smith, and R. Tuler. A pipeline architecture for factoring large integers with the quadratic sieve algorithm. *SIAM J. Computing*, 17(2):387–403, April 1988.

[129] W. L. Price. Annontated bibliographies of cryptography. Published as National Physical Laboratories technical reports since 1978.

[130] M. Rabin. *Digitalized Signatures as Intractable as Factorization*. Technical Report MIT/LCS/TR-212, MIT Laboratory for Computer Science, January 1979.

[131] M. Rabin. *How to exchange secrets by oblivious transfer*. Technical Report TR-81, Harvard Aiken Computation Laboratory, 1981.

[132] M. Rabin. Probabilistic algorithms for testing primality. *J. Number Theory*, 12:128–138, 1980.

[133] M. Rabin. Probabilistic algorithms in finite fields. *SIAM J. Computing*, 9:273–280, May 1980.

[134] M. O. Rabin. Digitalized signatures. In Richard A. DeMillo, David P. Dobkin, Anita K. Jones, and Richard J. Lipton, editors, *Foundations of Secure Computation*, pages 155–168, Academic Press, 1978.

[135] H. Riesel. *Prime Numbers and Computer Methods for Factorization*. Birkhäuser, Boston, 1985.

[136] R. L. Rivest and A. Shamir. How to expose an eavesdropper. *Communcations of the ACM*, 27:393–395, April 1984.

[137] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communcations of the ACM*, 21:120–126, February 1978.

[138] S. Sanders. Data privacy: what Washington doesn't want you to know. *Reason*, 24–37, January 1981.

[139] M. Santha and U. V. Vazirani. Generating quasi-random sequences from slightly-random sources. In *Proceedings of the 25th IEEE Symposium on Foundations of Computer Science*, pages 434–440, IEEE, Singer Island, 1984.

[140] R. Schroeppel and A. Shamir. A $TS^2 = O(2^n)$ time/space tradeoff for certain NP-complete problems. In *Proc. 20th Annual IEEE Symposium on Foundations of Computer Science*, pages 328–336, IEEE, San Juan, Puerto Rico, 1979.

[141] A. Shamir. How to share a secret. *Communcations of the ACM*, 22:612–613, November 1979.

[142] A. Shamir. On the cryptocomplexity of knapsack schemes. In *Proc. 11th ACM Symposium on Theory of Computing*, pages 118–129, ACM, Atlanta, 1979.

[143] A. Shamir. On the generation of cryptographically strong pseudo-random sequences. In *Proc. ICALP*, pages 544–550, Springer, 1981.

[144] A. Shamir. A polynomial-time algorithm for breaking the basic Merkle-Hellman cryptosystem. In *Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science*, pages 145–152, IEEE, Chicago, 1982.

[145] A. Shamir, R. L. Rivest, and L. M. Adleman. Mental poker. In D. Klarner, editor, *The Mathematical Gardner*, pages 37–43, Wadsworth, Belmont, California, 1981.

[146] C. E. Shannon. Communication theory of secrecy systems. *Bell Sys. Tech. J.*, 28:657–715, 1949.

[147] A. Sherman. *Cryptology and VLSI (a two-part dissertation).* PhD thesis, MIT EECS Dept, October 1986. Published as MIT Laboratory for Computer Science Technical Report MIT/LCS/TR-381 (Oct. 1986).

[148] G. J. Simmons. Cryptology. In *The New Encyclopædia Brittanica*, pages 860–873, Encyclopædia Brittanica, 1989. (Volume 16).

[149] G. J. Simmons, editor. *Secure Communications and Asymmetric Cryptosystems.* Volume 69 of *Selected Symposia*, AAAS, 1982.

[150] G. J. Simmons. Symmetric and asymmetric encryption. *ACM Computing Surveys*, 11:305–330, 1979.

[151] N. J. A. Sloane. Error-correcting codes and cryptography. In D. Klarner, editor, *The Mathematical Gardner*, pages 346–382, Wadsworth, Belmont, California, 1981.

[152] R. Solovay and V. Strassen. A fast Monte-Carlo test for primality. *SIAM J. Computing*, 6:84–85, 1977.

[153] U. V. Vazirani. Towards a strong communication complexity theory, or generating quasi-random sequences from two communicating slightly-random sources. In *Proc. 17th ACM Symposium on Theory of Computing*, pages 366–378, ACM, Providence, 1985.

[154] U. V. Vazirani and V. V. Vazirani. Efficient and secure pseudo-random number generation. In *Proceedings of the 25th IEEE Symposium on Foundations of Computer Science*, pages 458–463, IEEE, Singer Island, 1984.

[155] U. V. Vazirani and V. V. Vazirani. Trapdoor pseudo-random number generators, with applications to protocol design. In *Proceedings of the* 24*th IEEE Symposium on Foundations of Computer Science*, pages 23–30, IEEE, Tucson, 1983.

[156] J. von Neumann. Various techniques for use in connection with random digits. In *von Neumann's Collected Works*, pages 768–770, Pergamon, 1963.

[157] F. W. Winterbotham. *The Ultra Secret.* Futura, London, 1975.

[158] A. D. Wyner. An analog scrambling scheme which does not expand bandwidth, part 1. *IEEE Trans. Inform. Theory*, IT-25(3):261–274, 1979.

[159] A. D. Wyner. An analog scrambling scheme which does not expand bandwidth, part 2. *IEEE Trans. Inform. Theory*, IT-25(4):415–425, 1979.

[160] A. D. Wyner. The wire-tap channel. *Bell Sys. Tech. J.*, 54:1355–1387, 1975.

[161] A. C. Yao. How to generate and exchange secrets. In *Proceedings of the* 27*th IEEE Symposium on Foundations of Computer Science*, pages 162–167, IEEE, Toronto, 1986.

[162] A. C. Yao. Protocols for secure computations. In *Proceedings of the* 23*rd IEEE Symposium on Foundations of Computer Science*, pages 160–164, IEEE, Chicago, 1982.

[163] A. C. Yao. Theory and application of trapdoor functions. In *Proceedings of the* 23*rd IEEE Symposium on Foundations of Computer Science*, pages 80–91, IEEE, Chicago, 1982.

[164] G. Yuval. How to swindle Rabin. *Cryptologia*, 3:187–189, July 1979.