This work was partially funded by the National Science Foundation under grant numbers MCS-8002447, MCS-8204246, and MCS-8305382.

Provable Security of Cryptosystems: a Survey

Dana Angluin and David Lichtenstein
Yale University
YALEU/DCS/TR-288

October 1983

Abstract

This survey describes the recent explosion of results concerning rigorous proofs of the security of encryption, signatures, pseudo-random number generators, coin flipping, the oblivious transfer, and other cryptographic protocols, relative to assumptions on the computational difficulty of certain problems. Background material is included for the relevant number theoretic problems.

Table of Contents

1. Introduction

By now the proposal of Diffie and Hellman for public-key cryptography [24, 25], and the encryption and signature systems of Rivest, Shamir, and Adleman [60] and of Merkle and Hellman [47] are familiar to most readers. Lempel [42] gives an excellent survey of the developments in this area circa 1978. Other useful references are [18, 23, 41, 48].

However, these early public-key cryptosystems had a significant theoretical drawback: there was no proof of the security of the system. In the case of the Merkle-Hellman knapsack scheme, Shamir [63] has given a polynomial time algorithm for breaking the 1-iteration version of the modular knapsack, casting doubt also on the more complex versions. The RSA scheme is still believed to be secure, but while it *seems* that one would have to be able to factor quickly to break it, there is no proof of this.

There are good reasons why it is difficult to give proofs of the security of public key cryptosystems. As we shall detail below, for reasonable definitions of public key systems, a proof of security would imply that $P \neq NP$, a notoriously hard open problem in complexity theory. Thus it is only prudent to attempt proofs of security relative to some plausible assumption, e.g., that some problem like factoring is computationally difficult.

Another defect of the original proposals for public key cryptosystems is that even if the problem of completely deciphering an encrypted message is difficult, there is no guarantee of the security of partial information. For example, in a proposal to use a cryptosystem to play poker over the telephone, the ability to identify correctly the color of an opponent's card might confer a significant advantage, even if the value of the card could not be completely decrypted.

The work described below begins to provide more satisfactory solutions to these problems and others. Relative to assumptions about the computational difficulty of certain number theoretic problems, new cryptosystems have been proposed for which security of even partial information can be rigorously stated and proved, and there are now elegant schemes for generating pseudorandom bit sequences that are provably difficult to predict or to distinguish from truly random bit sequences. There are the beginnings of a theory of the adequacy and security of more complex cryptographic protocols, based on limited assumptions about the underlying cryptosystem. New and subtle relationships between complexity classes and the existence of one-way functions have been discovered.

All of this suggests that there are deep and important connections between computational complexity, security of communication, authentication, randomness, and information, that challenge us to explore and elucidate them.

2. Preliminaries on complexity and number theory

2.1. Complexity

Ideas from computational complexity play an integral role in public-key cryptography and variants of it. In public-key encryption, each participant A has a published fast encryption algorithm E_A and a secret fast decryption algorithm D_A. Given the encryption of a message $E_A(M)$, it is always possible to enumerate all possible messages, apply the published algorithm E_A until the message is found that encrypts to $E_A(M)$, and thereby discover the message M. However, possibility does not equal feasibility, and the effectiveness of a public-key system depends on the degree to which it is computationally infeasible to retrieve a message or significant information about a message from its encryption.

Two notions introduced by Diffie and Hellman [24] for the understanding of a public-key cryptosystem are a *one-way* function and a *trap-door* function. Intuitively, a function $f(x)$ is a one-way function if it is one to one and "easy" to compute, while its inverse $f^{-1}(y)$ is "hard" to compute.

A trap-door function is a somewhat more complex notion, since it is really a family of functions $f(k,x)$ for different values of k. It should have the following properties. For each value of k, $f(k,x)$ is a one to one function of x. It is "easy" to compute $f(k,x)$ given k and x. It is "hard" to compute x such that $y = f(k,x)$ given y and k.

In addition, there should be a "trap-door", which works as follows. There exists a function $d(k)$ and a function $g(k',y)$ such that whenever $y = f(x,k)$ then $x = g(d(k),y)$. Moreover, $g(k',y)$ is "easy" to compute given k' and y. Thus, the information $d(k)$ is a "trap-door" making the inversion of $f(x,k)$ easy. (Clearly, if this is to be true, the trap-door information $d(k)$ had better not be "easy" to compute from k.)

A one-way function would be useful for storing a password file; a trap-door function (with some additional properties) would be suitable for constructing a public-key encryption

system. Candidates for one-way and trap-door functions are described in the section on number theoretic functions.

The usual distinction of "easy" and "hard" in computational complexity is between problems that can be solved in time polynomial in the lengths of the inputs, and those that cannot. (For formal definitions of these notions, see for example the text of Hopcroft and Ullman [39].) However this distinction is not appropriate for cryptography for several reasons.

One difficulty is that a problem not solvable in polynomial time may be quite easy for most of its inputs, whereas an encryption system that is easy to break for all but a vanishing fraction of the message space is useless. Another is that an encryption system that with high probability can be broken by a fast probabilistic algorithm is similarly useless, even if there is no fast deterministic algorithm to break it. Still another difficulty is that computational complexity has generally been concerned with the order of growth of complexity of a uniform algorithm to solve the problem as a function of the length of the input. Since an encryption system is likely to be used with one or a few particular key lengths, a non-uniform attack requiring some modest built-in table of information might be a serious threat, even if it could not be generalized to a uniform algorithm. Also, the distinction of polynomial versus non-polynomial may be quite misleading, for example, while an algorithm with complexity n^{177} is "faster" than one with complexity $n^{loglog\ n}$, the crossover doesn't occur until $n = 2^{2^{177}}$.

As a concrete example of the pitfalls of computational complexity for cryptography, see Lempel's illustration of a protocol that is provably NP-hard to break, but which can be broken efficiently with high probability [42].

Thus, complexity notions must be applied with care in this field. Ideally, one would like an approach that could yield practical answers to questions such as: what fraction of messages of a given length can be broken by an adversary with certain computational resources? While no comprehensive approach has been developed, some of the above difficulties have been overcome by considering circuit complexity (a nonuniform measure) and probabilistic algorithms. Rather than the distinction of polynomial versus non-polynomial time (or circuit size), some authors have advocated proving general tradeoffs of the resources used versus the probability of breaking the system, but most of the results we

describe are not of this kind.

(See the excellent paper of Gill [33] for definitions and results relevant to probabilistic algorithms, including the definitions of the classes RP and BPP.)

2.2. Number theoretic problems

Many of the results we describe are based on properties of and assumptions about number-theoretic problems. Niven and Zuckerman [51] present a useful exposition of elementary number theory; for a treatment of the complexity of several number-theoretic problems, see [3]. In order to make the survey relatively self-contained on this point, we sketch some of the relevant facts.

The terms "easy" and "hard" will be used below to distinguish problems for which there are practically feasible algorithms yielding a high degree of confidence, from those for which there is no such practically feasible method for even a significant fraction of instances. Since these are not precisely defined terms, they will be left in quotes. More precise statements of the degree of "hardness" assumed of some of these problems will be given in subsequent sections describing specific results.

Primality testing and factoring

There is a probabilistic algorithm [57, 65] that takes a number N as input, runs in time polynomial in the length of the binary representation of N, and outputs 1 if N is prime, and outputs 0 with probability at least $1/2$ if N is composite. This algorithm may be re-run k times to increase the probability that it outputs 0 for a composite number to at least $1 - (1/2)^k$. Thus, testing a number for primality is generally taken to be an "easy" problem.

By contrast, the fastest probabilistic algorithms known for factoring do not appear to make it feasible to factor a number that is the product of two very large primes with any reasonable probability of success, so it is plausible to assume that factoring such numbers may be an intrinsically "hard" problem. (Of course, here we are generalizing from our own ignorance, a potentially dangerous course, particularly in science.)

Exponentiation and the discrete logarithm

For any number N, the set of numbers between 1 and $N-1$ (inclusive) that are relatively prime to N will be denoted $\mathbf{Z}_N^*$. $\mathbf{Z}_N^*$ forms a group under multiplication modulo N. Given an element a of $\mathbf{Z}_N^*$ and a positive integer x, the unique b in $\mathbf{Z}_N^*$ such that

$$b \equiv a^x \ (mod\ N)$$

may be computed quickly by the method of successive squaring. This problem is taken to be "easy".

If a and N are relatively prime, a variant of the greatest common divisor algorithm yields integers c and d such that

$$ca + dN = 1.$$

Thus, the unique b in $\mathbf{Z}_N^*$ congruent to c modulo N has the property that $ab \equiv 1\ (mod\ N)$. Thus, the problem of finding inverses in $\mathbf{Z}_N^*$ is "easy" as well.

If p is prime, then $\mathbf{Z}_p^*$ is a cyclic group of order $p-1$, and the elements that generate the group are called generators or primitive roots. Given a primitive root g of p and any element b of $\mathbf{Z}_p^*$, there is a unique value of x between 1 and $p-1$ such that

$$b \equiv g^x \ (mod\ p).$$

This value is called the *index* or *discrete logarithm* of b with respect to p and g. Finding the value of this x is the discrete logarithm problem. When $p-1$ consists entirely of small prime factors, there is a fast algorithm for the discrete logarithm problem [53], but otherwise, no fast algorithm is known, even if the factorization of $p-1$ is given. Thus, like the factoring problem, the discrete logarithm problem may plausibly be assumed to be computationally "hard".

Thus, exponentiation of a primitive root modulo a prime is a candidate for a one-way function, since it is "easy" and its inverse (the discrete logarithm) appears to be "hard".

Testing quadratic residuosity

An a in $\mathbf{Z}_N^*$ is said to be a *square* or *quadratic residue* modulo N if and only if the equation

$$x^2 \equiv a \ \ (mod\ N)$$

has some solution x. Any such solution is called a *square root* of a modulo N. Otherwise,

a is said to be a *nonsquare* or *quadratic nonresidue* modulo N. If p is a prime, then there is an efficient test to determine whether or not a is a square modulo p, namely, a is a square if and only if

$$a^{(p-1)/2} \equiv 1 \pmod{p}.$$

Half the elements of $\mathbf{Z}_p^*$ are squares, the other half are nonsquares.

By contrast, if N is the product of two large prime numbers, say p and q, there is no efficient algorithm known for determining whether a is a square modulo N, given a and N as input, but not the factorization of N.

In fact, using the Chinese Remainder Theorem it may be seen that a is a square modulo N if and only if it is a square modulo both p and q. Moreover, there is an easily computed quantity, the Jacobi symbol of a with respect to N, that is 1 in case a is either a square modulo both p and q or a nonsquare modulo both of them, while it is -1 in case a is a square modulo one of them and a nonsquare modulo the other. We note that half the elements of $\mathbf{Z}_N^*$ have Jacobi symbol 1, half have Jacobi symbol -1. Of elements of $\mathbf{Z}_N^*$ with Jacobi symbol 1, half are squares modulo N, and half are nonsquares.

The problem that would appear to be "hard" in this case is to distinguish squares from nonsquares modulo N among those a with Jacobi symbol 1 with respect to N. This is called the problem of *quadratic residuosity modulo a composite.*

Note that this problem becomes "easy" if we are given the factorization of N; we simply test whether a is a square modulo both p and q. Thus, this test has a "trap door", namely, the factorization of N.

Extracting square roots

What about the problem of actually finding the square roots of a quadratic residue modulo N? If p is prime, every square modulo p has exactly two square roots, and there is a probabilistic algorithm that runs in expected polynomial time to find them [1, 5].

However, if N is the product of two distinct odd primes, say p and q, then every square a modulo N has four square roots in $\mathbf{Z}_N^*$: x, y, $N-x$, and $N-y$, where x is not congruent to either y or $-y$ modulo N. Thus,

$$a \equiv x^2 \equiv y^2 \pmod{N},$$

so

$$N \mid (x^2 - y^2) = (x + y)(x - y),$$

which implies that p divides $(x + y)$ and q does not, or vice versa, since $N = pq$, and N does not divide either $(x + y)$ or $(x - y)$. Hence, if we take the greatest common divisor of N and $(x + y)$, we will factor N. Thus, if we can find the four square roots of a square modulo N, we can factor N. Conversely, if we are given the factorization of N into p and q, we can efficiently use the square root algorithm modulo the primes and the Chinese remainder theorem to find the four square roots modulo N.

Thus, squaring modulo a composite $N = pq$ is a candidate for a trap-door function. Given x and N, it is "easy" to compute x^2 modulo N. (This is a four to one rather than a one to one function; we must restrict the domain in some manner to make it one to one.) Given N and x^2 modulo N, it appears to be "hard" to find the four square roots of x^2. (In fact, it appears to be "hard" to find any of the square roots of x^2, as will be shown in section 2.3.) However, given x^2 and the factorization of N, it is "easy" to compute the four square roots.

Finding other roots and the RSA function

How hard is the problem of finding other roots modulo N, in other words, solving equations of the form $x^k \equiv b \pmod N$ given k, b, and N? Let $\phi(N)$ denote the number of elements of $\mathbf{Z}_N^*$.

If k is relatively prime to $\phi(N)$, then the mapping from x to x^k is one to one on $\mathbf{Z}_N^*$, and there exists an integer m such that $km \equiv 1 \pmod{\phi(N)}$. If $x^k \equiv b \pmod N$, then

$$b^m \equiv x^{km} \equiv x \pmod N$$

so the desired solution is $b^m \pmod N$. Thus, given a knowledge of the factorization of N, or of $\phi(N)$, it is "easy" to find m and, therefore, kth roots.

However, if the factorization and $\phi(N)$ are unknown, no efficient method is known for finding kth roots. Thus, the mapping of x to $x^k \pmod N$ is a candidate for a trap-door function, in which the trap door information is the factorization of N. The Rivest, Shamir, and Adleman (RSA) encryption method [60] is precisely this function, where $N = pq$ and k are the public information, and the factorization of N is kept secret.

2.3. Random self-reducibility

Why are these four problems: factoring, discrete logarithms, testing quadratic residuosity modulo a composite, and the RSA function, particularly suitable as bases for the cryptosystems to be described later? One reason is that the algebraic structure of this domain gives rise to a class of properties one might term *random self-reducibility*, which in turn can be used to show a problem is uniformly hard if it is hard at all. In particular, these reductions generally consist of taking a particular input and mapping it uniformly and randomly to other inputs in such a way that the answer for the original input can be recovered from answers for the targets of the random mapping.

For comparison, we demonstrate simple forms of random self-reduction for each of the problems discussed.

(The remainder of this subsection can be skipped without loss of continuity.)

Factoring

It was shown above that the ability to find the four square roots of a square modulo $N = pq$ allows us to find p and q. Suppose there exists an algorithm $A(N,a)$ that correctly finds one square root of a modulo N for some fraction ϵ of the squares a modulo N, when $N = pq$. We may construct another, probabilistic, algorithm $B(N)$ to factor its input N when $N = pq$ as follows.

B on input N randomly chooses a number r in the interval $[1, N-1]$. If r is not relatively prime to N, B halts and outputs the factorization of N. Otherwise (the typical case), B computes $b \in \mathbf{Z}_N^*$, where

$$b \equiv r^2 \pmod{N},$$

and calls $A(N,b)$. B then checks the answer x returned by A to see whether $x^2 \equiv b \pmod{N}$, i.e., whether A's answer is correct, and whether the greatest common divisor of $x + r$ and N is a proper divisor of N. If it is a proper divisor, B halts and outputs the factorization of N, otherwise it generates a new random number r and repeats this process.

The algorithm B outputs a correct factorization of N. At each iteration, it has a chance of at least $(1/2)\epsilon$ of factoring N, so its expected running time is at most $2/\epsilon$ times

the running time of A plus the overhead of checking the answer. Thus, if A is a fast algorithm and ϵ is large enough, B will be a reasonably fast algorithm for factoring N. Intuitively this says that if the problem of factoring the product of two large primes is "hard", then it cannot be "easy" to find even one square root of a nontrivial fraction of squares modulo such numbers. This proof is due to Rabin [56].

Discrete logarithm

Suppose there exists an algorithm $A(p,g,a)$ that correctly computes the discrete logarithm of a with respect to the prime p and the generator g for some fraction ϵ of the values of a in $\mathbf{Z}_p^*$. We construct another, probabilistic, algorithm $B(p,g,a)$ to compute the discrete logarithm correctly for all a in $\mathbf{Z}_p^*$. $B(p,g,a)$ calls $A(p,g,b)$ where

$$b \equiv g^r a \pmod{p}$$

for a randomly chosen value of r between 1 and $p-1$. (b ranges randomly over all of $\mathbf{Z}_p^*$.) When B gets an answer x from A, it checks to see whether

$$b \equiv g^x \pmod{p}.$$

If so, then this is one of the values where A works correctly, and B halts with output y, where y is the unique number in the interval $[1, p-1]$ such that

$$y \equiv x - r \pmod{p-1}.$$

Otherwise, B generates another random value of r and repeats this procedure. The output is the correct discrete logarithm for a, because

$$g^r a \equiv g^x \pmod{p},$$

so

$$a \equiv g^{x-r} \pmod{p}.$$

Thus, B is correct for all values of a, and the expected running time of $B(p,g,a)$ is $1/\epsilon$ times the running time of $A(p,g,a)$ plus the overhead of checking the answers that A gives. If A is a "fast" algorithm, and ϵ is "large enough", then B will be a relatively fast algorithm too. This reduction shows that if the discrete logarithm problem is "hard", it cannot have a nontrivial fraction of "easy" instances for each p and g.

The RSA function

This self-reduction is very similar to the one for the discrete logarithm, so it will only be sketched. Suppose $A(N,k,b)$ is an algorithm that computes $b^{1/k}$ $(mod\ N)$ for a fraction $\epsilon > 0$ of b in $\mathbf{Z}_N^*$ when $N = pq$ and k is relatively prime to $\phi(N)$. The algorithm $B(N,k,b)$ will repeatedly choose random numbers r in $\mathbf{Z}_N^*$ and call $A(N,k,br^k)$ until the value returned, say x, is correct. Then the kth root of b is just $r^{-1}x$ $(mod\ N)$, so B outputs this and halts.

Whenever k is relatively prime to $\phi(N)$, the mapping from r to r^k $(mod\ N)$ is one to one, and the expected running time of B will be $1/\epsilon$ times the running time of A, plus the calling overhead.

Testing quadratic residuosity modulo a composite

This problem is different from the preceding three in that the answers given by a putative quadratic residuosity testing algorithm cannot be easily checked if the quadratic residuosity problem is hard. Thus, the assumptions and approach must be somewhat different. To simplify the exposition, we consider a slightly modified form of the quadratic residuosity problem, in which a nonresidue is also given as input.

Suppose there exists an algorithm $A(N,y,a)$ such that whenever N is the product of two primes and y is a nonresidue with Jacobi symbol 1 with respect to N, A correctly answers whether or not a is a quadratic residue modulo N for a fraction $1/2 + \epsilon$ of the possible values of a in $\mathbf{Z}_N^*$ that have Jacobi symbol 1 with respect to N.

We construct an algorithm $B(N,y,a)$ for the quadratic residuosity problem as follows. B chooses m random numbers $r_1, r_2, ..., r_m$ from the interval $[1, N-1]$. (The value of m will be specified later.) If any of the values r_i is not relatively prime to N, then B can split N, and in the case that $N = pq$, B has the factorization of N, and can compute and output whether a is a quadratic residue modulo N.

Otherwise (the typical case), B generates a random sequence of m bits: $d_1, d_2, ..., d_m$, computes $b_i \in \mathbf{Z}_N^*$ such that

$$b_i \equiv a r_i^2 y^{d_i} \ (mod\ N),$$

and calls $A(N,b_i)$ for $i = 1,2,...,m$.

Note that when y is a quadratic nonresidue, if $d_i = 0$, then b_i is a quadratic residue if and only if a is, and if $d_i = 1$, then b_i is a quadratic residue if and only if a is not. Thus, each answer from A can be interpreted as a "vote" for whether a is a residue or not, depending on the value of d_i and the answer from A. If a majority of the votes from A are that a is a quadratic residue, then B outputs "quadratic residue", otherwise it outputs "quadratic nonresidue".

Provided y is a nonresidue with Jacobi symbol 1, for each i, every element of Z_N^* with Jacobi symbol 1 is equally likely to be b_i, so each call of A has a probability of $1/2 + \epsilon$ of yielding a correct "vote" about the residuosity of a. By Chebyshev's inequality, the probability of a majority of the m calls yielding correct votes is at least $1 - \delta$ provided $m \geq 1/(4\delta\epsilon^2)$.

Thus, by running for time proportional to $1/(\delta\epsilon^2)$ times the running time of A plus the calling overhead, $B(N,y,a)$ can achieve a probability of $1 - \delta$ of being a correct residuosity test whenever $N = pq$, y is a nonresidue with Jacobi symbol 1 modulo N, and a in Z_N^* has Jacobi symbol 1. Hence, if the problem of testing residuosity given a nonresidue of Jacobi symbol 1 is "hard", then there is no fast algorithm that gives the correct answer on $1/2 + \epsilon$ of the values for a "nontrivial" fraction ϵ.

2.4. Hard bits

Another significant property of these functions is the existence of "hard bits" for them, that is, an efficient reduction of the problem of computing the function to deciding a related boolean predicate. For example, inverting the RSA function is efficiently reducible to deciding whether x is odd or even, given $x^k \ (mod\ N)$.

Such reductions have been used in the construction of pseudo-random number generators as secure as the basic function, using the boolean predicate to generate the bits of the sequence (see section 4). They may also be used in attacks on public-key systems based on these functions, to show that a communication convention that leaks information about the boolean predicate may be used to compromise the encryption system (see section 3.2).

We describe some of the results related to hard bits in the remainder of this subsection.

Blum and Micali [10] show that computing the discrete logarithm with respect to the generator g modulo the prime p is efficiently reducible to the problem of deciding whether the index of x modulo g and p is less than $(p-1)/2$. The reduction works as follows.

Given x, we may efficiently test whether x is a square modulo p by computing $x^{(p-1)/2} \ (mod\ p)$. If it is a square, then we know that the index of x is even, that is, the low order bit of the index of x is 0. Otherwise, the low order bit is 1, and we may set it to 0 by multiplying x by g^{-1} modulo p.

The resulting quantity, say y, is now a square in either case, and we may find its two square roots z and $-z$ in expected polynomial time [1, 5]. One of these, namely the one whose index is less than $(p-1)/2$, has an index that is the index of y divided by 2. We use the oracle to decide which of z and $-z$ has index less than $(p-1)/2$, and repeat this procedure with that square root to determine the next lowest order bit of the index of x, and so on, until the quantity becomes 1, when all the bits of the index of x are determined.

In fact, Blum and Micali show a much stronger result, that if there is an oracle that correctly answers the question of whether the index of x is less than $(p-1)/2$ for a fraction $1/2 + \epsilon$ of the values of x in $\mathbf{Z}_p^*$, where $\epsilon > 0$, for all p and g, then there is a probabilistic algorithm that correctly computes the discrete logarithm for all p and g and runs in expected polynomial time.

Blum, Blum, and Shub [9] use a hard bit based on testing quadratic residuosity to construct a pseudo-random sequence generator. Suppose $N = pq$ where p and q are primes congruent to 3 modulo 4, so that x and $-x$ have the same Jacobi symbol modulo N. Then any square modulo N has four square roots, exactly one of which is itself a square modulo N. Suppose there is an oracle that for any square modulo N gives the parity of the unique square root that is also a square modulo N. Then to test whether x with Jacobi symbol 1 is a square modulo N, we submit x^2 to the oracle. If its answer agrees with the parity of x, then x is a square modulo N, otherwise, $-x$ is. Blum, Blum, and Shub show that the problem of testing quadratic residuosity modulo integers N of this form is efficiently reducible to determining the parity of the unique square root of a square that is also a square.

Goldwasser, Micali, and Tong [36] show that inverting the RSA function is efficiently

reducible to deciding each of the following three properties of x, given $x^k \ (mod\ N)$.

1. the parity of x,
2. whether $x \leq N/2$,
3. the value of any of the m least significant bits of x, where the low-order $m+1$ bits of N are 011 .. 11.

Goldwasser, Micali, and Tong also show that factoring $N = pq$ (where p and q are both congruent to 3 modulo 8 or both congruent to 7 modulo 8) is efficiently reducible to deciding each of the following two properties, given $x^2 \ (mod\ N)$.

1. whether the parity of y is equal to the parity of z, where y and z in $\mathbf{Z}_N^*$ are the two distinct square roots of $x^2 \ (mod\ N)$ that are less than $N/2$,
2. the parity of y, where y is the unique square root of $x^2 \ (mod\ N)$ that both is less than $N/2$ and has Jacobi symbol 1 modulo N.

Ben-Or, Chor, and Shamir [4] show that inverting the RSA function is efficiently reducible to an "interval oracle" that decides, given $x^k \ (mod\ N)$, whether x is in some nonnegligible interval I of $\mathbf{Z}_N^*$, (that is, the length of I divided by N is bounded below by ϵ and above by $1 - \epsilon$, for some $\epsilon > 0$.) They also show that an oracle that guesses the least significant bit of x from $x^k \ (mod\ N)$ with an error probability of less than $1/4 - \epsilon$ is sufficient to construct a probabilistic algorithm that inverts the RSA function in expected polynomial time. Other results are given for single RSA bits, and a bit as hard as factoring.

Long and Wigderson [43] argue that if a one-way function $f(x)$ exists, then there must be more than $O(log\ |x|)$ bits of x that are hard to compute from $f(x)$. They consider the problem of proving that a number of bits are "simultaneously hard" and show that computing the discrete logarithm is efficiently reducible to computing any non-constant boolean predicate of $O(log\ |p|)$ bits of the index of x with respect to g and p, where $|p|$ is the length of the binary representation of p. They also consider the case of an oracle for such a predicate that makes errors.

3. Provable security of cryptographic systems

The beginnings of the current, more refined analysis of the security of cryptographic systems were provided by Rabin [56] when he introduced a signature system based on finding square roots modulo a composite number, and formally related its security to the difficulty of factoring.

In Rabin's scheme, the signer publishes a composite number N, a hashing function C, and a number b. The factorization $N = pq$ is kept secret. To sign the message M, the signer randomly generates a string U of k bits, applies the hashing function C to MU to obtain a number c between 1 and N, and checks to see whether the equation

$$c \equiv x(x + b) \pmod{N}$$

has a solution. If so, the signed message consists of M, U, and x. Otherwise, the signer repeats this procedure, choosing a new value of U. The signature is checked using the equation

$$x(x + b) \equiv C(MU) \pmod{N}.$$

Rabin shows that solving $c \equiv x(x + b) \pmod{N}$ is equivalent to extracting square roots modulo N. Although the roles of C and U are not formally analyzed, Rabin shows that the ability to extract square roots of $1/R$ of the squares modulo N in time $F(N)$ allows one to factor N in expected time $RF(N) + 2\log N$ steps. (This is essentially the argument given in section 2.3.)

The guarantee that if the problem (in this case, forging signatures) is "easy" for a substantial fraction of instances, then factoring is "easy" seems like a strong one, but Goldwasser and Micali [34] point out that the ability to forge signatures on a vanishingly small fraction of strings of some length (e.g., ASCII representations of English sentences, broken up into pieces of appropriate length) could be a security weakness if those strings were the probable ones.

Lipton gives another objection to the kinds of arguments heretofore used concerning cryptographic protocols. He shows that the scheme proposed by Shamir, Rivest, and Adleman [64] for mental poker using the RSA encryption system may "leak" partial information about the values of the cards, even though an adversary may not be able to invert the encryptions completely.

3.1. Probabilistic encryption

The first solution to the problems of skewed distributions and partial information was the encryption system proposed by Goldwasser and Micali [34, 35] based on the problem of deciding whether a number is a quadratic residue modulo a composite. (See section 2.2 for definitions.) Their encryption scheme works as follows.

For each user A there is public information consisting of a composite number N_A that is the product of two primes, and a number y_A that is a quadratic nonresidue with Jacobi symbol 1 modulo N_A. A keeps secret the factorization of N_A.

To send a message consisting of the bit string $b_1 b_2 ... b_k$ to A, another user successively encrypts the bits by choosing a random quadratic residue modulo N_A if b_i is 1, and a random quadratic nonresidue with Jacobi symbol 1 otherwise. The encryption of the bit string is the resulting sequence of residues and nonresidues.

Knowing the factorization of N_A, A can efficiently test quadratic residuosity and thus read the encrypted message. To produce a random quadratic residue, the other user simply chooses a random number and squares it modulo N_A; to produce a random nonresidue, the random square is multiplied by y_A modulo N_A. It is assumed that there is some fixed polynomial bound on the length of messages that will be sent as a function of the length of the composite N_A.

Notice that this encryption scheme is *probabilistic*, that is, there are a large number of different encryptions of a given message, and one is chosen at random by the sender to represent the message. Thus it is reasonable to consider the encryption of a single bit (which doesn't make sense if the encryption function is deterministic.) Goldwasser and Micali show that under the appropriate assumption about the difficulty of testing quadratic residuosity, this scheme is "polynomially secure" and also "semantically secure".

The assumption made about testing quadratic residuosity is that if $C_{k,\epsilon}$ is the minimum size of any circuit $C(b,N)$ that correctly computes quadratic residuosity of b with respect to N for all $b \in Z_N^*$ for a fraction at least ϵ of the composite numbers $N = pq$ of length k, then for any $\epsilon > 0$, $C_{k,\epsilon}$ is not bounded by any polynomial in k.

The definition of polynomial security involves the notions of a line-tapper and a message-finder. Intuitively, a line-tapper is a collection of circuits, one for each key length,

that take as input N, y, and an encryption of a message of suitable length and produce as output a 0 or a 1. For a given line-tapper, a message-finder is a collection of circuits, one for each key length, such that for every key length, for a nontrivial fraction of N and y corresponding to that key length, the message-finder circuit produces two messages such that the line-tapper has detectably different acceptance percentages for encryptions of the two messages. The definition of polynomial security is that for no polynomially bounded line-tapper does there exist a polynomially-bounded message-finder.

The definition of semantic security concerns probability distributions on the messages sent by the users. It is assumed that each user randomly generates messages according to a fixed probability distribution specific to that user. A "meaning" is any function defined on messages; the problem of the adversary is to guess the meaning $f(m)$ of a message, given some encryption of m. For a given user A, and for each key length and associated message length, there is some "most probable" meaning value v, depending on the message distribution of A. Let p_k denote the probability of this value.

Roughly, semantic security is the nonexistence of a polynomially bounded collection of circuits $\{M_k\}$ that correctly compute the value $f(m)$ from encryptions of m with probability distinguishably greater than p_k when m is drawn according to the distribution for A, for a fixed fraction of keys of length k.

In [35], Goldwasser and Micali give general definitions of a cryptographically strong predicate, the construction of a cryptographically strong public key encryptions system from such a predicate, and proofs that any cryptographically strong public key encryption system is polynomially secure, and any polynomially secure public key encryption system is semantically secure. In addition to the system described above based on the quadratic residuosity problem, they also demonstrate a probabilistic encryption system based on the second of the two bits as hard as factoring described in section 2.4. This is a polynomially secure system under the appropriate assumption about the computational difficulty of factoring.

Yao [67] also abstracts from the specific number theoretic problems used, and gives very general definitions and theorems concerning ways in which secure public-key encryption systems may be constructed from trapdoor functions which are only very "weakly" one-way. A key lemma is that if there is enough uncertainty about a one-bit

predicate on the inverse of a weakly one-way function, then by taking the exclusive-or of a large but polynomially bounded number of instances of the predicate, the uncertainty may be made nearly 1/2.

3.2. Adversaries other than passive eavesdroppers

The results above concern security against a *passive eavesdropper*, that is, an adversary who is assumed to be able to intercept all the encrypted communications in the system, and who has access to the published encryption algorithms. (Note that such an adversary may generate pairs consisting of plaintext and cyphertext from chosen plaintext for himself in a public-key scheme.) Goldwasser, Micali, and Tong [36] consider attacks by adversaries with other kinds of capabilities. *Malicious users* are defined to be adversaries that have the capabilities of passive eavesdroppers, and also are legitimate users of the cryptosystem, so that they may read, store, and send encrypted messages. *Meddlers* are defined to be adversaries with a great deal of control over the communication system -- in addition to the capabilities of malicious users, they may alter, delete, delay, and resend messages between other users. (Dolev and Yao [27] consider security against meddlers, see section 7.1.)

Goldwasser, Micali, and Tong demonstrate the vulnerability of the RSA, Rabin, and Goldwasser and Micali cryptosystems to attacks by malicious users. For example, they show that given an oracle to determine the parity of x from $x^e \ (mod\ N)$ it is computationally easy to invert the RSA function. Such an oracle can be introduced by the communication conventions, if for example, "yes" is represented by encoding an even number, "no" is represented by encoding an odd number. Suppose A is a malicious user who wants to determine the parity of x for some value $x^e \ (mod\ N_B)$. A waits until B asks him a question, and then replies $x^e \ (mod\ N_B)$. From B's subsequent behavior, A determines whether the answer was interpreted as "yes" or "no", and concludes that x is even or odd accordingly. Given enough of this kind of information, A can decrypt messages sent to B.

Goldwasser, Micali, and Tong describe other single-bit oracles equivalent to inverting RSA and to factoring, and show how they could be introduced by communication conventions. A slightly different type of problem is exhibited for the Goldwasser and

Micali probabilistic encryption scheme. Suppose that the communication system occasionally garbles messages, and there is a provision for asking for a re-transmission in this case. Then a malicious user A can get information about quadratic residuosity modulo N_B as follows. To test whether x is a residue modulo N_B, A probabilistically encodes some innocuous message, like "Shall we go to lunch now?", using B's published information. Then, for a random collection of the quadratic residues in the encryption, A multiplies in the value of x. The result will be either an unchanged message (if x is a quadratic residue modulo N_B), or a garbled message (if x is a nonresidue.) Then whether B asks for a re-transmission or not allows A to conclude that x is a nonresidue modulo N_B or not. Enough of this kind of information allows A to decrypt messages sent to B.

Goldwasser, Micali, and Tong ascribe these problems to the fact that these are public-key cryptosystems, that is, all the communications to B use the same key, so that the communications between a malicious user A and a (non-malicious) user B can leak information about communications between some other user C and B. They propose instead a private key system (in which every pair of communicating users has a different key) in which the private keys can be established publically.

The Goldwasser, Micali, Tong scheme works as follows. Each user A has a published number N_A, a product of two primes, that he uses to authenticate his communications with other users. A knows the factorization of N_A, but other users do not. When A and B wish to communicate, they contact one another and make use of their primary keys N_A and N_B to authenticate to each other their secondary keys S_{AB} and S_{BA}, which are each the product of two primes. (The authentication procedure is described separately below.) Initially, A knows the factorization of S_{AB} and B does not. The two secondary keys are congruent to 1 modulo 4 so that -1 has Jacobi symbol 1.

Once they have authenticated their secondary keys to one another, A factors S_{AB} for B -- this is done by B choosing a number y with a Jacobi symbol -1 modulo S_{AB} and authenticating $y^2 \ (mod\ S_{AB})$ to A. A replies by authenticating a square root of y^2 with Jacobi symbol 1 modulo S_{AB} to B. The two roots B now has allow him to factor S_{AB}. A then sends the encryption of a seed for a cryptographically secure pseudo-random number generator to be used to simulate a one-time pad. The symmetrical procedure is carried out for S_{BA}. (The next section will decribe cryptographically secure pseudo-random number

generators and their use to simulate one-time pads.)

The sub-protocol for A to authenticate a message X to B is as follows. They jointly flip a coin to determine random numbers r until one is found such that rX is a square modulo N_A, at which point A sends one square root, w, of this quantity to B. B checks that $w^2 \equiv rX \pmod{N_A}$. (The coin-flipping protocol is considered elsewhere; it is essential that neither A nor B be able to influence the outcome.)

Goldwasser, Micali, and Tong claim that the resulting communication system is provably secure against malicious users. They show that it is not secure against meddlers, and pose as an open question whether there is any cryptosystem provably secure against meddlers. An added bonus of their scheme is that since it is a simulated one-time pad, the ratio of the number of bits transmitted to the number of bits encoded approaches one, unlike the probabilistic encryption schemes described above, in which each bit is encoded by a separate element of $\mathbf{Z}_N^*$.

4. Pseudo-random number generators

The probabilistic encryption scheme of Goldwasser and Micali, and many other cryptographic protocols that have been proposed, call for the generation of random numbers. The customary way of simulating the generation of random numbers in an implementation is to use a pseudo-random number generator, that is, a deterministic algorithm that takes a small number of (ideally) truly random bits and "expands" them into a longer sequence of bits that has some properties of randomness.

What properties of randomness are desirable? Knuth [40] gives several statistical tests that can be used to judge the suitability of a proposed pseudo-random number generator. However in the context of cryptography, a stronger requirement is necessary: an adversary should find it difficult to predict missing terms in the sequence, given a substantial fraction of the terms in the sequence.

As an example of this kind of requirement, consider the use of pseudo-random sequences in the efficient simulation of a one-time pad, as described by Shamir [62]. Two users, each with knowledge of a random seed k, and using the same pseudo-random number generator, can encode and decode messages by computing the bitwise exclusive-or of the cleartext or cyphertext with a given segment of the pseudo-random number sequence. It

would be desirable to prove that no eavesdropper, even one who knows the cleartexts of some of the messages and the structure of the pseudo-random number generator, can "easily" infer the contents of unknown messages or the identity of the seed.

Plumstead [52] has shown that the traditional linear congruential pseudo-random number generator is insecure in this sense. Suppose that a pseudo-random sequence is generated according to the rule

$$X_i \equiv aX_{i-1} + b \ (mod\ N).$$

Suppose further that an adversary has access to the successive terms of the sequence as they are generated, but does not know a, b, or N. Then there is an efficient algorithm that will permit the adversary to predict the next term on the basis of the preceding terms with a total of only $2 + log\ N$ errors of prediction.

Shamir [62] was the first to propose a pseudo-random sequence generator that is *cryptographically strong*, i.e., is such that an adversary given some subset of the numbers does not find it materially easier to compute any of the missing ones. Shamir's pseudo-random number generator is based on the difficulty of inverting the RSA encryption scheme, and works as follows.

The originator of the sequence publishes N, S, and a sequence of keys which are relatively prime to $\phi(N)$: $K_1, K_2, ..., K_t$. (A segment of odd primes, e.g., 3, 5, 7, 11, ... is suggested for the keys.) The factorization $N = pq$ is kept secret. Then, the sequence of pseudo-random numbers is $R_1, R_2, ..., R_t$, where $R_i \equiv S^{1/K_i} \ (mod\ N)$ for each i.

Shamir shows that the problem of determining R_1 given N and S is not substantially harder than the problem of determining R_1 given N, S, and $R_2, R_3, ..., R_t$, where l and the key sequence are fixed. In particular, there is a polynomial $P(l,n)$ such that if $C(l,n)$ is a circuit that solves the second problem for a fraction $g(N)$ of the values of the seed modulo each N of length n, then then there is a circuit of size $P(l,n)$ plus the size of $C(l,n)$ to solve the first problem for a fraction $g(N)$ of the seeds modulo each N of length n. Since the first problem is the problem of inverting the RSA encryption scheme for the fixed value K_1, this gives confidence in this scheme relative to the difficulty of inverting RSA encryption.

However, Shamir does not show that no adversary can easily compute, say, the parity or approximate order of magnitude of missing numbers. In the use envisioned by Shamir of

these numbers as a one-time pad, an eavesdropper could conceivably compute some useful partial information about the relevant cleartexts, even though he could not invert the cyphertexts.

A solution to the problem of partial information for pseudo-random sequence generators was first found by Blum and Micali [10], who propose a generator based on the discrete logarithm problem. A simpler generator based on the quadratic residuosity problem is given by Blum, Blum, and Shub [9]. Later work by Yao [67] gives a definition of an abstract one-way function and a method of using any such function to construct a pseudo-random sequence generator. Goldwasser, Micali and Tong describe a generator based on the difficulty of factoring [36]. (This last reference contains errors in the proofs.)

Each of these generators may be understood as an instance of the following general format for a pseudo-random sequence generator.

For each seed length n, a pseudo-random sequence generator is a state machine. It consists of a set Y of states, each of which is represented by a string of length n, and two functions, f and b. f is the next state function, and b is a function that computes the output, either 0 or 1, from a given state. The randomly chosen seed y_0 of the generator serves as the start state for the machine. Since the number of states is, in general, exponential in the length of the output, it is infeasible to represent the machine explicitly, but it is enough to specify f and b. The length of the output is $L = n^k$ for some fixed k.

The pseudo-random sequence corresponding to the seed y_0 is $b_0, b_1, \ldots, b_{L-1}$ where $b_i = b(f^i(y_0))$.

It is assumed that the next state function f is the inverse of a one-way function and that b is easy to compute. Thus, given a state y of the generator, it is easy to generate the output, but hard to compute the next state. Because of this, the sequence is actually generated in reverse order: randomly choose not the start state y_0, but rather the last state, y_{L-1}. Then compute y_{i-1} by applying f^{-1} to y_i, storing the bits $b_i = b(y_i)$ as they are computed, for $1 \leq i \leq L-1$. (Since f is the inverse of a one-way function, f^{-1} is easy to compute.) Finally, output the sequence $b_0, b_1, \ldots, b_{L-1}$.

The proofs of security involve showing that computing f reduces to the problem of predicting the next output bit, given the current state of the generator. Thus, the ability

to predict the next bit contradicts the assumption that f is hard to compute.

Notice that after one has output the first l bits, one can publish f, y_0, and y_{l-1}, and anyone can verify that the bits were indeed output according to the generator, although it is still hard to compute the next output bit.

For the Blum and Micali generator, the states are of the form $<x,p,g>$ where p is a prime, g is a generator for p, and x is in $\mathbf{Z}_p^*$. The value of $f(x,p,g)$ is $<w,p,g>$, where w is the discrete logarithm of x with respect to p and g. The predicate $b(x,p,g)$ is 1 if $x < p/2$ and 0 otherwise. Blum and Micali show that computing this predicate of the next state of the generator, given the current state, with a significant statistical advantage is as hard as computing the discrete logarithm; this result is described in the section on hard bits, section 2.4.

The Blum, Blum, and Shub generator is based on the problem of quadratic residuosity. The states of the generator are of the form $<x,N>$, where $N = pq$, p and q are primes congruent to 3 modulo 4, and x is an element of $\mathbf{Z}_N^*$ that is a square modulo N. The value of the next state function $f(x,N)$ is $<y,N>$, where y is the unique square root of x that is also a square modulo N. The predicate $b(x,N)$ is the parity of x. Blum, Blum, and Shub show that computing b applied to the next state, given the current state, with any significant statistical advantage is as hard as computing quadratic residuosity modulo N. (The proof in the deterministic case is given in the section on hard bits, section 2.4.)

The proofs in [9, 10] were extended by Yao [67], first to show that, given any set of bits of either generator, one cannot hope to guess any missing bit (rather than just the last bit). He then strengthened his result to show that, assuming the difficulty of the underlying problem (discrete logarithm or quadratic residuosity), no random polynomial time test can distinguish pseudo-random sequences from truly random ones. This property seems to be the analog in the domain of feasible computation of Martin-Lof's results [45] on infinite sequences that pass all effective statistical tests.

A corollary of Yao's result is that as long as one keeps all information about the states secret, one can output the bits in the order in which they were computed. In such a case, the next state function can be a one-way function, rather than the inverse of a one-way function. Another very interesting corollary of Yao's results is that if pseudo-random

sequence generators exist, they may be used to simulate random polynomial time computations in deterministic time 2^{n^ϵ} for any $\epsilon > 0$. This surprising result is further described in section 6.

The Blum, Blum, and Shub generator has the additional property that $f^i(y)$ is computable in time polynomial in the relevant parameters, including $|i|$ (even though i itself may be exponentially large). In [9] it is suggested that the seed y can therefore be used as a compact way of storing a table of passwords, where an account name can be used as an index into the seqence of bits, and its password can be the 25 bits $b_{name}, \ldots, b_{name+24}$. However, it has not yet been proved that the generator remains secure when bits are given to an adversary from exponentially distant sections of the sequence.

It would also be interesting to analyze how well the above technique does as a hash code -- what are the chances of collisions, and how does it compare with the hash functions studied by Carter and Wegman [15]?

5. Other cryptographic problems

5.1. Digital signatures

Both the Rabin [56] and RSA [60] schemes allow signatures of messages, so that a user can not only encrypt a message to make it unreadable to all except one user, but can also sign a message, so that any user can tell that only one user could have sent it. (The Rabin scheme is described in section 3.)

Typically, a message is signed by applying a hard to compute but easy to invert function f to the message. (These functions are generally the inverses of the easy to compute but hard to invert functions used in encryption.) In this way, if f^{-1} is a trapdoor function, only someone with knowledge of the trapdoor information could be expected to compute the signature of a given message, but anyone could confirm that signature, merely by computing f^{-1} on the signature.

As with encryption, f may be hard to compute in general, but easy to compute on ASCII representations of English words or on some other sparse set of strings. Another problem is that, even if f is everywhere hard to compute, anyone can produce signed random numbers simply by picking the signature at random and then computing f^{-1} of

the signature to see what it is that has been signed. Many of the protocols discussed in other sections require the ability to sign randomly chosen numbers, so this last problem can not be avoided by the claim that the chance of finding the signature of an English sentence is negligible. A third problem is that one must be able to prove that knowledge of some (polynomially-bounded) number of previously signed messages does not help in forging a new signed message.

Goldwasser, Micali, and Yao [37] propose two signature schemes that overcome these problems, relative to assumptions of the difficulty of factoring or inverting the RSA function. In the first scheme, the signer publicizes a composite $N = pq$ where both p and q are congruent to 3 modulo 4, a randomly chosen number y with Jacobi symbol -1 modulo N, and a randomly chosen x_0 in $\mathbf{Z}_N^*$. The factorization of N is kept secret.

To sign a message m consisting of k bits $b_1 \ldots b_k$, the signer computes a sequence of k square roots as follows. If x_0 is not a square, it is multiplied by the appropriate one of -1, y, or $-y$ to make it a square. Then, if b_1 is 0, then x_1 is set to the unique square root of this square that has Jacobi symbol 1 and is less than $N/2$. If b_1 is 1, then x_1 is set to the unique square root of this square that has Jacobi symbol -1 and is less than $N/2$. This process is repeated on x_1 and b_2 to obtain x_2, and so on, until x_k is obtained. The signed message consists of m and x_k.

The signer is able to compute the signature efficiently because he knows the factorization of N and can test quadratic residuosity and extract square roots modulo N. Other users may verify the signature because they can compute squares and Jacobi symbols modulo N without knowing the factorization of N. Actually, the scheme must be somewhat modified in order to allow the signer to sign many messages. Rivest suggested the following tree-structured scheme, with $c \geq 2$. The signer initially signs (using x_0) a list of $2c$ random elements of $\mathbf{Z}_N^*$, the first c of which are used to sign messages as above (one message per point), and each of the last c of which is used to sign a list of $2c$ additional random elements of $\mathbf{Z}_N^*$, to be used in the same way. Some convention must be adopted to prevent proper initial segments of signed messages from being legal signed messages.

The security that Goldwasser, Micali, and Yao prove for this scheme is that if there is a polynomial time adversary that can produce one additional signed message, given the published information of the signer and a polynomial quantity of signed messages, with

probability at least some $\epsilon > 0$, then there is a probabilistic algorithm to factor composites of the above kind in expected time polynomial in the length of the composite and ϵ, for any probability distribution on the messages to be signed.

Note that the number of bits of overhead for signatures is quite reasonable in this scheme, since it is essentially two numbers from $\mathbf{Z}_N^*$ per signed message, one to sign the message, and one to sign a list of new signature points.

The second scheme described by Goldwasser, Micali, and Yao uses Shamir's idea for a pseudo-random number generator based on the difficulty of inverting the RSA encryption function ([62], see section 4.) The signer publishes a composite number $N = pq$, a number t between $N/2$ and N, and a randomly chosen number x from $\mathbf{Z}_N^*$. He keeps the factorization of N secret.

The sequence of consecutive primes starting with t is used for forming signatures. To sign a message m consisting of k bits, the signer allocates the next $2k+9$ unallocated consecutive primes p_i greater than t for this message, and then computes and provides $x^{1/p_i} \pmod N$ for a certain set S of these primes according a coding scheme roughly as follows. The first three and the last six primes in the block are used to form "begin" and "end" markers. The remaining $2k$ primes are divided into k pairs, and each pair is used to code a single bit of the message m. If the bit is 0, the first of the two primes in the pair is included in S; if it is a 1, the second of the two primes is included.

Goldwasser, Micali, and Yao claim that this signature scheme is secure in a sense similar to that for the first scheme, basically because to produce a new signature entails producing a new value $x^{1/p_i} \pmod N$, which Shamir has shown is essentially as difficult as inverting the RSA scheme, even in the presence of a polynomial number of other such values. Note that this scheme has considerably more signature overhead, since for each bit of the signed message at least one number from $\mathbf{Z}_N^*$ is given.

5.2. Coin flipping and the oblivious transfer

A number of more complex protocols involve fair coin flips or oblivious transfers, for example, the protocol for authentication of a message proposed by Goldwasser, Micali, and Tong [36], described in section 3.2. A *fair coin flip* is a two-party protocol that determines one of two equally likely outcomes, "heads" or "tails". The outcome is known to both

parties after the protocol is completed. There is some kind of guarantee that if either party follows the protocol and the other does not then either "cheating" will be detectable or the probabilities of the outcomes will not be significantly altered. An *oblivious transfer* is also a protocol for two parties, A and B, which allows A to transfer a "secret" to B with probability 1/2. After the completion of the protocol, B knows whether or not the secret was successfully transferred to him, but A does not know. There is similarly some sort of guarantee about the detection of "cheating".

Rabin invented the concept of the oblivious transfer, and proposed the following implementation. A chooses a composite $N = pq$. The secret to be transferred (or not) is a factor of N. A sends N to B, who chooses a number x and sends $x^2 \ (mod\ N)$ to A. A finds the four square roots x, $-x$, y, and $-y$ of x^2 modulo N, picks one of the four at random and sends it to B. The idea is that A has no way of knowing which of the square roots of x^2 B has, and half of A's choices will allow B to find a factor of N, while the other half provide B no new information.

However, Fischer [31] has pointed out the following difficulty with this scheme. It is conceivable that there is a "special" set of squares such that B can easily generate an element of this set without knowing any of its square roots. Then the argument that in half the cases A is providing no new information to B breaks down, since B now always gets a square root he did not have previously. The informal correctness arguments failed to eliminate this possibility. Micali and Rackoff [49] have proposed a modification of the basic Rabin protocol that essentially entails an "interactive proof" by B that he knows one of the roots of x^2, without revealing any information about which one.

Blum [8] describes several applications of the ideas of Rabin's oblivious transfer to coin-flipping, the exchange of secrets, and certified mail. He also describes a coin flip based on factoring that appears to avoid the above problem, as follows. A chooses a composite $N = pq$ and sends N to B. B chooses $x < N/2$ and sends $x^2 \ (mod\ N)$ to A, who replies with a guess of "larger" or "smaller". B then sends x to A, who replies with the factorization of N. The outcome is "heads" if A guessed "larger" and x is in fact the larger of the two square roots of x^2 that are less than $N/2$, or if A guessed "smaller" and x is the smaller of the two. Since A sends only a one bit guess to B, the claim that he is not providing B with any new information may be formally justified. Note that B can

influence whether x is "larger" or "smaller" and potentially take advantage of any bias in A's guesses. A variant of this coin flip is used by Goldwasser, Micali, and Tong in their authentication procedure [36].

A coin flip based on quadratic residuosity may be obtained from Goldwasser and Micali's results on probabilistic encryption [34]. A chooses a composite $N = pq$ and a number x from $\mathbf{Z}_N^*$ with Jacobi symbol 1 with respect to N, and sends them both to B. B guesses "residue" or "nonresidue", and A replies with the factorization of N, allowing B to verify whether his guess was correct or not.

For each of the pseudo-random bit sequence generators that follow the "state" paradigm of section 4, there is a related coin flip. In particular, A sends a choice of a state y to B, who replies with a guess of the predicate b applied to the next state $f(y)$. A then reveals $f(y)$ to B, who can verify the outcome since f^{-1} and b are easily computed. (Since f is in general hard to compute, A generates the pair $<y,f(y)>$ by choosing $z = f(y)$ at random and letting $y = f^{-1}(z)$.) This leads to coin flips as secure as the discrete logarithm, using the Blum and Micali generator [10], and as secure as quadratic residuosity, using the Blum, Blum, and Shub generator [9].

The problem of giving a general useful formal specification of the correctness of a coin flip or oblivious transfer protocol appears to be a difficult one. (The reader will note that we were studiedly vague in the introduction to this section.) Current axiomatic approaches, for example that of Even, Goldreich, and Lempel [28], while appealing, seem to be quite incomplete. For example, they give an abstract implementation of the oblivious transfer assuming a secure public key cryptosystem and a secure conventional cryptosystem. However, in their protocol, B can always arrange to get the secret if the public key encryption and decryption functions happen to commute for different keys -- an unlikely possibility that is nonetheless not excluded in the axiomatic treatment.

DeMillo, Lynch, and Merritt [22] describe an abstract coin flip and prove its security within their framework; this is further described in section 7.1.

5.3. Protocols for other problems

Built on top of these basic systems are various more complex protocols, to accomplish such tasks as signing contracts [7, 8, 11, 28, 30, 55, 56], establishing private keys [36, 46], secret ballots [16, 22], mental poker [34, 64], protecting proprietary software [21], certified mail [8, 11, 58], sharing a secret [61], exchanging secrets [7, 8, 44], computing $f(i,j)$ without revealing i and j [66], computer system security [2, 20], database security [19, 59], anonymity in the use of electronic mail [16], and monitoring arms treaties [17]. There are a number of interesting ideas in these more complex protocols, but they are beyond the scope of this survey.

6. Relations to complexity theory

We do not know whether one-way or trap-door functions exist, but there has been progress in determining the consequences of their existence for complexity theory. Intuitively speaking, a function f is one-way if f is easy to compute, but its inverse is difficult to compute.

Brassard [12] has investigated one way of formalizing the definition of a one-way function. A function f from strings to strings is defined to be *nice* if it is injective, computable in polynomial time, has its domain in NP and range in co-NP, and is such that there exists a polynomial $p(n)$ such that whenever x is in the domain of f, $|x| \leq p(|f(x)|)$. (This last condition insures that the output of f^{-1} is of length polynomial in the input.)

Examples of nice functions are exponentiation of a primitive root modulo a prime (whose inverse is the discrete logarithm) and multiplication of primes (whose inverse is factoring).

A nice function is defined to be *weakly one-way* if and only if its inverse f^{-1} is not computable in polynomial time. Brassard shows that if there exists a nice weakly one-way function then P is a proper subclass of NP $\cap$ co-NP, so in particular, P $\neq$ NP. One possible way to give evidence that a nice function might be weakly one-way would be to show that f^{-1} is NP-hard. However, Brassard also shows that if there exists a nice function whose inverse is NP-hard, then NP = co-NP.

Even and Yacobi [29] extend this idea and show that an extension of the hypothesis that NP $\neq$ co-NP implies that if f is computable in polynomial time and is injective, then

f^{-1} is not NP-hard. In another paper [13], Brassard exhibits a recursive f such that f^{-1} is hard to compute, even given an oracle for f.

Brassard [14] gives a setting for the asymptotic analysis of a variant of public-key cryptography he calls *transient key cryptography*, in which the key and message are guaranteed to be about the same length, and the key is only used once. He discusses the need for results concerning general time bounds (instead of polynomial versus nonpolynomial) and defines the notion of a transient key cryptosystem that achieves a level of security $t(n)$ -- no randomized algorithm that runs in time $t(n)$ can expect to decode more than an exponentially vanishing fraction of encoded messages of a given length. He defines transient key cryptosystems relative to an oracle, and demonstrates the existence of a relativized transient key cryptosystem that achieves a level of "nearly" $2^{n/\log n}$ security.

Yao [67] considers two other definitions of a one-way function. One may be used to construct a "perfect" pseudo-random number generator (see section 4); the other is a *strong one-way function*. Roughly speaking, f is a strong one-way function if it is bijective on some subset of $\{0,1\}^*$ and there exists a probabilistic polynomial time computable sequence of probability distributions $p_1, p_2, p_3, \ldots$, invariant with respect to f, such that the smallest circuit C_n that successfully inverts f with probability greater than $1/2$ when the inputs are drawn according to p_n has size exceeding any polynomial in n for all sufficiently large n.

Yao then claims the following exciting theorem: if there exists a strong one-way function, then

$$\mathrm{RP} \subset \bigcap \{\mathrm{DTIME}(2^{n^\epsilon}) \mid \epsilon \geq 0\}.$$

(RP is the class of sets recognizable in random polynomial time, that is, by polynomial time bounded probabilistic Turing machines that accept with probability at least $1/2$ if they accept at all. See Gill's paper [33] for formal definitions and results.)

Thus, for example, the existence of a strong one-way function would imply the existence of a better deterministic algorithm for testing primality than any currently known, since the composites are in RP.

The idea of the proof is that any problem in RP would distinguish random from pseudo-random sequences if it could be solved using true random bits but not pseudo-random bits. Assuming that a strong one-way function exists, it may be used to construct

a pseudo-random number generator that cannot be distinguished by any probabilistic polynomial time algorithm from a truly random sequence. Thus, the pseudo-random number generator can be used in place of true random numbers in the simulation of RP computations with no difference in the outcomes.

What problems are candidates for being strong one-way functions? Yao claims that the first of the following assumptions is enough to guarantee that factoring will yield a strong one-way function, and the second is sufficient in the case of the discrete logarithm.

1. Let $G(n)$ be the smallest size of any circuit to factor 4/5 of all $2n$-bit numbers that are the products of two n-bit primes. Assume that $G(n)$ exceeds any polynomial in n for all sufficiently large n.
2. Let $L(n) = max\ \{L_{p,g} \mid log(p+1) \leq n\}$, where $L_{p,g}$ is the smallest size of any circuit to compute the discrete logarithm with respect to generator g modulo the prime p. Assume that $L(n)$ exceeds any polynomial in n for all sufficiently large n.

(Note that the second assumption concerns the worst-case complexity for circuits for the discrete logarithm that have a knowledge of p and g "built in".)

The above results point to the difficulty of proving the existence of one-way functions, since it would entail stronger results about complexity classes than we currently are able to prove. However, it might be reasonable to try to prove the existence of some version of one-way functions relative to some plausible complexity assumption, for example, that NP ∩ co-NP contains sets that are not in BPP. No such results are known at present.

(BPP is the class of sets S such that there is some polynomial time bounded Turing machine M that for every input either accepts with probability greater than 3/4 or accepts with probability less than 1/4, and S is the set of inputs M accepts with probability greater than 3/4. See Gill's paper [33] for definitions and results concerning BPP.)

7. Relative correctness of protocols

Protocols typically involve a sequence of message exchanges between two or more parties using a particular cryptosystem. The correctness and security of a protocol may involve protection against cheating by the parties, protection against passive eavesdroppers

or active saboteurs, and may provide for arbitration procedures in the case of a dispute between the parties about their adherence to the protocol or its outcome. Protocols are generally no more secure than the underlying cryptosystem, and may be less so. One approach to the analysis of more complex protocols is to attempt to prove their security relative to appropriate assumptions about the security of the basic operations used.

7.1. Algebraic approaches to security

Dolev and Yao [27] have initiated a kind of algebraic approach to the security of protocols relative to the security of the underlying cryptosystem. They consider adversaries who are assumed to have the capabilities of legitimate users of the protocols, and also are able to intercept messages between any two users of the protocols and possibly substitute their own messages.

As an example of the kinds of problems they wish to detect, consider the following protocol for A to send a message to B. A sends the triple $(A, E_B(M), B)$ to B, and B replies with $(B, E_A(M), A)$. An adversary Z can read the message M simply by intercepting the message $(A, E_B(M), B)$ and sending the message $(Z, E_B(M), B)$ to B, who will reply with $(B, E_Z(M), Z)$, allowing Z to read M. This is a case in which Z reads a message supposed to be secret to him without cracking the underlying cryptosystem, because of weaknesses introduced by the protocol.

One fix for this is to require signatures on the encrypted messages. The message M signed by A is denoted MA. The new protocol is for A to send $(A, E_B(MA), B)$ to B, who replies with $(B, E_A(MB), A)$. This scheme is shown to be secure in the relative sense defined by Dolev and Yao.

A further elaboration of this scheme is to encrypt the message before signing it, and then to encrypt it again. Thus, A sends $(A, E_B(E_B(M)A), B)$, and B replies with $(B, E_A(E_A(M)B), A)$. However, Dolev and Yao show this to be insecure as follows. Z intercepts the message $(A, E_B(E_B(M)A), B)$ and sends to B the message $(Z, E_B(E_B(M')Z), B)$, where $M' = E_B(M)A$ B replies with $(B, E_Z(E_Z(M')B), Z)$, allowing Z to recover the encryption $E_B(M)$ from M'. Z now sends to B the message $(Z, E_B(E_B(M)Z), B)$, and B's reply allows him to read M.

Dolev and Yao define *cascade* protocols, which allow applications of encryption and

decryption functions, and *name-stamp* protocols, which also allow the application and removal of signatures. The notion of security they define for cascade protocols is essentially whether a message M that is supposed to be secret is obtainable by some party from the messages and operations available to him in the free algebra over the decryption and encryption functions, with the cancellation rule that the product of A's encryption and decryption functions may be cancelled if it occurs in either order. A similar setting is used for name-stamp protocols.

Dolev and Yao give characterization results for the security of cascade protocols, and an $O(n^8)$ algorithm to test the security of name-stamp protocols. Dolev, Even, and Karp [26] improve the latter algorithm to $O(n^3)$.

This approach has been greatly extended and refined by DeMillo, Lynch, and Merritt [22]. They give set-theoretic models of the information available to and the capabilities of each party to a protocol. In their setting, the value of some predicate is *hidden* from a party A if there exist two models, both consistent with the information available to A, in one of which the predicate is true and in the other false.

The basic model is extended to cover stochastic protocols, in which replies are chosen with certain probabilities. For a given user and a given set of outcomes S, the *defensive weight* of S is defined to be the probability of arriving at any outcome from S if he follows his stochastic protocol against an (omniscient) adversary who always chooses so as to maximize the probability of arriving at an outcome from S. Thus, the stochastic and "hidden knowledge" aspects of security are not fully integrated. For example, for the coin flip they describe, the guarantee to one of the parties is that heads and tails each have defensive weight 1/2, while for the other party the defensive weight of both outcomes is 1, and the relevant guarantee is that the knowledge to influence the outcome is hidden from the first party.

DeMillo, Lynch, and Merritt give formal definitions and proofs of the correctness and security of protocols for secret ballots, signatures, coin flipping, and oblivious transfer. They also show a certain problem cannot be solved, namely, totally safe hostage exchange.

These algebraically based definitions of security are useful for proving a minimal kind of safety for more complex protocols, to avoid the kinds of pitfalls exemplified above.

However, this approach gives a fairly weak notion of security, since the possible actions of the eavesdropper or saboteur are restricted to composing known encryption and decryption functions with known messages, or, in general, to performing one of a known, limited, set of operations. Also, security in this sense does not rule out partial information about the secret messages.

7.2. Random cypher models

Hellman, Karnin, and Reyneri [38] propose a model of a random cypher: the encryption functions E_k are random permutations of the message space, and the cryptanalyst has access to them by evaluating $E_k(m)$ and $D_k(m)$ for chosen values of k and m. In this model, the cryptanalyst's chance of correctly guessing the key k even in a chosen plaintext attack is very small unless he asks questions about a large fraction of all possible keys.

A related model is described by Yao in [66], in which the encryption and decryption functions are randomly chosen permutations of the set of all strings of length n, and each party has access to an oracle for his own encryption and decryption permutations and the encryption permutations of the others. Protocols are probabilistic algorithms, and the number of oracle calls used by each party is bounded by a polynomial in n. Thus, each party can only evaluate the permutations on a negligible portion of their domains. Definitions and theorems are given relative to this model of a cryptosystem for the security of protocols for two parties A (which has the secret value i) and B (which has the secret value j) to cooperate to compute some function $f(i,j)$ without revealing much more information about i and j than the value of $f(i,j)$.

8. Remarks

The progress in provable security sketched in the preceding sections is truly exciting; however, it is important to look as well at the problem areas that remain to be addressed.

One obvious point is the overwhelming reliance on number theoretic problems. For example, if factoring should prove to be probabilistically solvable in polynomial time even for a significant subset of composites of the form $N = pq$, the protocols based on the difficulty of quadratic residuosity, the RSA function, and factoring may be severely or

fatally compromised. On the other hand, there are practical objections to systems based on arithmetic modulo very large numbers as too slow for applications involving very much data.

Both considerations suggest that we should search for other types of computational problems to serve as bases for cryptographic protocols. Yao [67] has made a start in this direction by giving abstract constructions of secure pseudo-random sequence generators and encryption systems from functions with certain properties. To gain some greater assurance that the types of functions required actually exist, it would be desirable to prove their existence relative to the weakest complexity assumption possible.

One problem area already mentioned is the attempt to move away from the distinction of polynomial versus nonpolynomial toward more general resource/success tradeoffs.

Another area that has not received very much theoretical attention is the larger systems environment of the proposed protocols. What sorts of guarantees must there be on provisions for establishing public keys, for disavowing compromised signatures, and various other "meta-functions" in order for a particular protocol to remain secure? See [6, 32, 50, 54] for some interesting criticisms and issues along these lines.

The last problem area is a related one, namely, a methodology for a modular approach to the construction and verification of the correctness and security of more complex protocols out of simpler ones. Most of the rigorous proofs of relative security that have been given are for fairly simple protocols, but the proofs themselves are rather complicated. Proofs about more complex protocols, or interacting systems of protocols, will probably require a modular approach even to be feasible. As part of this effort, it would be useful to have general formal specifications of the security and correctness of protocols for various cryptographic problems.

9. Acknowledgements

This survey developed out of a seminar in the Computer Science Department at Yale in the fall of 1982. We would like to thank all the students in the seminar for their help: Josh Cohen, Steve Gold, Ming Kao, Jingke Li, Tim Peierls, Lenny Pitt, Sid Porter, Yoav Shoham, and Greg Sullivan. Conversations with Manuel Blum, Shafi Goldwasser, Doug

Long, Michael Merritt, Silvio Micali, and Andy Yao were invaluable to our understanding of the material. Without Michael Fischer's enthusiasm and interest in cryptography, this survey would not have come to be. A preliminary draft of the survey was part of a proposal to the National Science Foundation submitted by the authors and Michael Fischer and Dan Gusfield. We would like to thank the anonymous referees of the proposal, as well as Josh Cohen, Dan Gusfield, Neil Immerman, and Evangelos Kranakis for their comments and suggestions on previous drafts. We would welcome any further comments and suggestions from our readers.

References

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

[2] Akl, S. G. and Taylor, P. D.
Cryptographic solution to a multilevel security problem.
Technical Report, Queen's Univ. Dept. of Comp. and Info. Sci. TR-82-142, 1982.

[3] Angluin, D.
Lecture notes on the complexity of some problems in number theory.
Technical Report, Yale Univ. Dept. of Computer Science, Report # 243, 1982.

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

[5] Berlekamp, E. R.
Factoring polynomials over large finite fields.
Math. Comp. 24:713-735, 1970.

[6] Berson, T. A. and Bauer, R. K.
Local network cryptosystem architecture.
In *Proc. IEEE CRYPTO 81 workshop*, pages 138-143. Univ. of Calif. at Santa Barbara ECE report 82-04, 1982.

[7] Blum, M.
How to exchange (secret) keys.
ACM Trans. on Computer Systems 1:175-193, 1983.

[8] Blum. M.
Three applications of the oblivious transfer.
1982.
Preprint.

[9] Blum, L., Blum, M., and Shub, M.
A simple secure pseudo-random number generator.
1982.
Presented at IEEE CRYPTO 82 workshop on Communication Security, Santa Barbara, CA. Preprint.

[10] Blum, M. and Micali, S.
How to generate cryptographically strong sequences of pseudo random bits.
In *Proc. 23rd IEEE Symp. on Foundations of Computer Science*, pages 112-117. IEEE, 1982.

[11] Blum, M. and Rabin, M. O.
Digital mail certification.
1981.
Preprint.

[12] Brassard, G.
A note on the complexity of cryptography.
IEEE Trans. on Inform. Theory IT-25:232-233, 1979.

[13] Brassard, G.
Relativized cryptography.
In *Proc. 20th IEEE Symp. on Foundations of Computer Science*, pages 383-391. IEEE, 1979.

[14] Brassard, G.
A time-luck tradeoff in cryptography.
J. Comp. Sys. Sci. 22:280-311, 1981.

[15] Carter, J. L. and Wegman, M. N.
Universal classes of hash functions.
In *Proc. 9th ACM Symp. on Theory of Computing*, pages 106-112. ACM, 1977.

[16] Chaum, D.
Untraceable electronic mail, return addresses and digital pseudonyms.
Comm. ACM 24:84-88, 1981.

[17] Chaum, D.
Silo watching.
In *Proc. IEEE CRYPTO 81 workshop*, pages 138-139. Univ. of Calif. at Santa Barbara ECE report 82-04, 1982.

[18] Chaum, D., Rivest, R. L., and Sherman, A. T., eds.
Advances in Cryptology: Proceedings of Crypto 82.
Plenum Press, 1983.

[19] Davida, G., Wells, D., and Kam, J.
A database encryption scheme with subkeys.
ACM Trans. on Database Systems 6:312-328, 1981.

[20] DeMillo, R. and Lipton, R.
A system architecture to support a verifiably secure multilevel security system.
In *Proc. 1980 IEEE Symp. on Security and Privacy, Berkeley, CA*, pages . IEEE, 1980.

[21] DeMillo, R., Lipton, R., and McNeil, L.
Proprietary software protection.
In *Foundations of Secure Computation*, pages 115-129. Academic Press, 1978.

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

[23] Denning, D. E.
Cryptography and Data Security.
Addison-Wesley, 1982.

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

[25] Diffie, W. and Hellman, M. E.
Multiuser cryptographic techniques.
In *Proc. AFIPS 1976 NCC*, pages 109-112. AFIPS Press, Montvale, N. J., 1976.

[26] Dolev, D., Even, S., and Karp, R. M.
On the security of ping-pong protocols.
1982.
Presented at IEEE CRYPTO 82 workshop on Communication Security, Santa Barbara, CA. Preprint.

[27] Dolev, D. and Yao, A. C.
On the security of public key protocols.
In *Proc. 22nd IEEE Symp. on Foundations of Computer Science*, pages 350-357. IEEE, 1981.

[28] Even, S., Goldreich, O., and Lempel, A.
A randomized protocol for signing contracts.
Technical Report, Technion, C. S. Dept. TR-233, 1982.

[29] Even, S. and Yacobi, Y.
An observation concerning the complexity of problems with few solutions and its applications to cryptography.
Technical Report, Technion, C. S. Dept. TR-167, 1980.

[30] Even, S. and Yacobi, Y.
Relations among public key signature systems.
Technical Report, Technion, C. S. Dept. TR-175, 1980.

[31] Fischer, M. J.
Remark at Lenny Pitt's qualifying exam.
1982.

[32] Gasser, M.
Limitations of encryption to enforce mandatory security.
In *Proc. IEEE CRYPTO 81 workshop*, pages 130-134. Univ. of Calif. at Santa Barbara ECE report 82-04, 1982.

[33] Gill, J.
Computational complexity of probabilistic Turing machines.
SIAM J. Comput. 6:675-695, 1977.

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

[35] Goldwasser, S. and Micali, S.
Probabilistic encryption.
1982.
preprint.

[36] Goldwasser, S., Micali, S., and Tong, P.
Why and how to establish a private code on a public network.
In *23rd IEEE Symp. on Foundations of Computer Science*, pages 134-144. IEEE, 1982.

[37] Goldwasser, S., Micali, S., and Yao, A.
Strong signature schemes.
In *15th ACM Symp. on Theory of Computing*, pages 431-439. ACM, 1982.

[38] Hellman, M. E., Karnin, E. D., and Reyneri, J.
On the necessity of cryptanalytic exhaustive search.
In *Proc. IEEE CRYPTO 81 workshop*, pages 2-5. Univ. of Calif. at Santa Barbara ECE report 82-04, 1982.

[39] Hopcroft, J. E. and Ullman, J. D.
Introduction to Automata Theory, Languages, and Computation.
Addison-Wesley, 1979.

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

[41] Konheim, A. G.
Cryptography: a primer.
John Wiley and Sons, 1981.

[42] Lempel, A.
Cryptology in transition.
Computing Surveys 11:285-303, 1979.

[43] Long, D. and Wigderson, A.
How discreet is the discrete log?
In *15th ACM Symposium on Theory of Computing*, pages 413-420. ACM, 1983.

[44] Luby, M., Micali, S., and Rackoff, C.
How to simultaneously exchange a secret bit by flipping a symmetrically-biased coin.
1983.
FOCS 83, to appear.

[45] Martin-Lof, P.
The definition of random sequences.
Inform. and Contr. 9:602-619, 1966.

[46] Merkle, R.
Secure communications over insecure channels.
Comm. ACM 21:294-299, 1978.

[47] Merkle, R. and Hellman, M. E.
Hiding information and receipts in trapdoor knapsacks.
IEEE Trans. on Inform. Theory IT-24:525-530, 1978.

[48] Meyer, C. H. and Matyas, S. M.
Cryptography: a New Dimension in Computer Data Security.
John Wiley and Sons, 1982.

[49] Micali, S. and Rackoff, C.
In preparation.
1983.

[50] Needham, R. M. and Schroeder, M. D.
Using encryption for authentication in large networks of computers.
Comm. ACM 21:993-999, 1978.

[51] Niven, I. and Zuckerman, H. S.
An Introduction to the Theory of Numbers.
John Wiley and Sons, Inc., 1966.

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

[53] Pohlig, S. and Hellman, M.
An improved algorithm for computing logarithms over GF(p) and its cryptographic significance.
IEEE Trans. on Info. Theory IT-24:106-110, 1978.

[54] Popek, G. J.and Kline, C. S.
Encryption procedures, public key algorithms, and digital signatures in computer networks.
In *Foundations of Secure Computation*, pages 133-153. Academic Press, 1978.

[55] Rabin, M. O.
Digitalized signatures.
In *Foundations of Secure Computation*, pages 155-168. Academic Press, 1978.

[56] Rabin, M. O.
Digital signatures and public key cryptosystems intractable as factoring.
Technical Report, MIT LCS TR-212, 1979.

[57] Rabin, M. O.
Probabilistic algorithm for testing primality.
J. Number Theory 12:128-138, 1980.

[58] Rabin, M. O.
Transaction protection by beacons.
Technical Report, Harvard Univ. Aiken Comp. Lab. TR-29-81, 1981.

[59] Rivest, R. L., Adleman, L., and Dertouzos, M. L.
On data banks and privacy homomorphisms.
In *Foundations of Secure Computation*, pages 169-177. Academic Press, 1978.

[60] Rivest, R. L., Shamir, A., and Adleman, L.
A method for obtaining digital signatures and public key cryptosystems.
Comm. ACM 21:120-126, 1978.

[61] Shamir, A.
How to share a secret.
Technical Report, MIT LCS TM-134, 1979.

[62] Shamir, A.
On the generation of cryptographically secure pseudo-random sequences.
In *Proc. Eighth Colloquium on Automata, Languages, and Programming*, pages 544-550. Springer-Verlag, Lecture Notes in Comp. Sci. No. 115, 1981.

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

[64] Shamir, A., Rivest, R., Adleman, L.
Mental poker.
Technical Report, MIT LCS TM-125, 1979.

[65] Solovay, R. and Strassen, V.
A fast Monte-Carlo test for primality.
SIAM J. on Comput. 6:84-85, 1977.

[66] Yao, A. C.
Protocols for secure computations.
In *Proc. 23rd IEEE Symp. on Foundations of Computer Science*, pages 160-164. IEEE, 1982.

[67] Yao, A. C.
Theory and applications of trapdoor functions.
In *Proc. 23rd IEEE Symp. on Foundations of Computer Science*, pages 80-91. IEEE, 1982.