Varieties of Secure Distributed Computing

Matthew Franklin
Columbia University

Moti Yung
IBM T. J. Watson Research Center

December 23, 1996

1 Introduction

Suppose that the heads of the Fortune 500 companies agree to cooperate in a comprehensive study of their collective financial health. The study might consist of statistical functions that combine existing raw data from each company, e.g., "the sum of 1993 net earnings of all 500 companies," or "the standard deviation of the average change in net profits over successive quarters" or "the average percentage of gross earnings spent on cryptographic research." Further suppose that, although all companies wish to know the results of this study, none is willing to risk the exposure of its individual financial data in the process. If an independent trusted agent is available (e.g., Price-Waterhouse), the problem is easy. Each company gives its raw data to the trusted agent, who computes the statistics on everyone's behalf. In the absence of an independent trusted agent, however, can these titans of industry still achieve their goal?

They can. Secure distributed computing (or "fault-tolerant distributed computing," or "oblivious circuit evaluation") is the problem of evaluating a function (or, equivalently, a circuit) to which each player has one secret input, such that the output becomes commonly known while the inputs remain secret. The problem would be trivial in the presence of a trusted agent. All players give their private inputs to the agent, who computes the function and distributes the output. The task of protocols for secure distributed computation can be considered to be the *simulation* of the presence of a trusted agent by a group of players who are mutually untrustworthy.

Of course, untrustworthiness, for cryptographers, is a many-flavored thing. It might mean that all parties behave perfectly during the protocol, but later some group of gossippers compare notes to try to learn everyone else's secrets. A stronger meaning might be that some fixed fraction of the parties cheat during the protocol, but in such a way that the outcome (i.e., the value of the function) remains the same. Even stronger would be if some fixed fraction of the parties cheated arbitrarily during the protocol; yet stronger adversarial behavior can be imagined.

In this paper, we will see solutions to the Fortune 500 problem (or any other computational problem) that assume nothing more than that each company trusts that there are at least 333 other companies that will not betray it (plus secure phone lines). Other solutions show that if conference-calling is also allowed, then each company need only assume that 250 other companies are honest. Still other solutions need only assume that the Chief Number Theorist of each company certifies that certain problems (such as quadratic residuosity) will remain intractable for as long as its financial information remains sensitive.

Results in the field can be divided into two main categories: protocols and complexity results. Protocols can be divided into two main categories: cryptographic and non-cryptographic. Cryptographic protocols can be divided into two main categories: two-party protocols and multi-party protocols. These are the lines along which the bulk of this paper is organized.

1.1 Historical Background

Secure distributed computing is a very general and powerful notion. It was preceded by numerous investigations of protocols for a variety of special-purpose (and interrelated) cryptographic tasks. Secret sharing [88] [90] is a multi-party protocol in which a designated player distributes a message for later recovery by some authorized subcollection of the remaining players; it is discussed in more detail in Section 2.1.6. Bit commitment, a two-player version of secret sharing, is discussed in Section 2.1.4. Coin flipping [23] is a two-party protocol that arrives at a common random bit; this was one of the earliest cryptographic protocols. Mental poker [89] [75] [43] [60] [96] [53] [44] [45] is a protocol for producing, and partially applying, a random permutation (i.e., shuffle and deal a deck of cards); as evidenced by the number of references, this problem was something of a critical test case for the cryptography community as the notion of security grew in sophistication over the years. Oblivious Transfer [23] [21] [29] is a fundamental two-party protocol that transfers a bit with uncertainty; it is discussed in more detail in Section 2.2.2. Secret exchange [25] [27] [76] [64] [95] is a two-party protocol that transfers a message in each direction with certainty; it is discussed in more detail in Section 2.4.4. Electronic money [32] [33] is a collection of protocols (e.g, withdrawal, purchase, deposit) that implement payment schemes without any physical requirements. Secret-ballot election schemes [42] [18] [65] are essentially a special case of secure computation in which the function is a simple sum of ones and zeros.

Early work in the formalization of cryptographic protocols is also of interest. One approach is algebraic [49] [77], proving security by relating actual protocols homomorphically to theoretically perfect protocols. A second approach is logical [31], proving security with respect to some reasonable axiomatization.

Lastly, we mention some general notions that are related to secure distributed computing. Instance-hiding schemes [1] [11] involve a weak computational agent exploiting one or more strong but untrusted computational agents to compute a function for secret inputs. When one player has a secret circuit and the other player has non-secret data (or vice versa), then the problem of secure distributed computing reduces to a "minimum-knowledge" (or "zero-knowledge") circuit simulation, solved in the general case by [56], and later improved by [69]. More will be said about zero-knowledge protocols in Section 2.1.5.

1.2 Organization of the Paper

The next section gives short descriptions of cryptographic and other primitives that are used in the upcoming protocols. Section 3 presents one model for secure distributed computation, as well as pointing out where "competing" models differ. Two-party cryptographic protocols are presented in Section 4, multi-party cryptographic protocols in Section 5, and multi-party non-cryptographic (sometimes called "unconditional") protocols in Section 6. Section 7 presents some of the complexity results and lower bounds that are known for these protocols, and Section 8 closes with some discussion and open questions.

2 Preliminaries

In this section, we give short overviews of cryptographic and distributed computing primitives that will be used in upcoming protocols: number theory basics, indistinguishable probability distributions, one-way functions, trapdoor functions, encryption, bit commitment, interactive and zero-knowledge proof systems, secret sharing, and instance hiding.

2.1 Number Theory Basics

In this subsection, we briefly review two common intractability assumptions upon which cryptographic security have been based: the Quadratic Residuosity Assumption and the Discrete Logarithm Assumption.

A quantity is "negligible" if it is smaller than the reciprocal of any polynomial of relevant parameters, and otherwise "non-negligible." When this term is used, the relevant parameters will usually not be stated explicitly. Negligibility can be used to formalize the "hardness" of a problem, a typical assumption made in cryptographic protocols.

Let n be the product of two primes p and q, each congruent to 3 modulo 4. Let a be an integer between 0 and $n-1$. Define $QRP_n(a)$ to be 0 if a is a square mod n, and 1 otherwise. The **Quadratic Residuosity Assumption** (QRA) states that no probabilistic polynomial-time Turing Machine can, on input n and a, output $QRP_n(a)$ non-negligibly better than random guessing (i.e., bounded above $\frac{1}{2}$ by the reciprocal of a polynomial of the size of n). However, there is an efficient algorithm for computing $QRP_n(a)$ if the factorization of n is known.

Let p be a prime, let g be a generator of $Z_p{}^*$ (i.e., $Z_p{}^* = \{g^i | i \in Z_p{}^*\}$), and let a be an integer between 0 and $p-1$. Define $DLP_{p,g}(a)$ to be i such that $g^i = x \bmod p$, $0 \leq i < p$. The **Discrete Log Assumption** (DLA) states that no probabilistic polynomial-time Turing Machine, can, on input p, g, and a, output $DLP_{p,g}(a)$ non-negligibly better than random guessing.

The quadratic residuosity function and the discrete logarithm function each has a property, "random self-reducibility," that will be used often in protocols described in this paper. Informally, a function has this property if it can be computed at any input from its value at "random-looking" inputs. More formally, f is random self-reducible if there are polynomial-time functions ϕ and $\sigma_1, \cdots, \sigma_k$ such that (1) if r is chosen uniformly at random from $\mathrm{dom}(f)$ then each $\sigma_i(x, r)$ is uniformly distributed over $\mathrm{dom}(f)$ for every $x \in \mathrm{dom}(f)$, and
(2) $f(x) = \phi(x, r, f(\sigma_1(x, r)), \cdots, f(\sigma_k(x, r)))$ for all $x, r \in \mathrm{dom}(f)$. The function QR_n is random self-reducible over Z_n^* (for any n), since (1) $xr \bmod n$ is a uniformly distributed element of Z_n^* whenever r is chosen at random from Z_n^* and (2) $QR_n(x) = QR_n(xr \bmod n) \oplus QR_n(r)$. The function $DLP_{p,g}(a)$ is random self-reducible over Z_p^* since (1) $ag^r \bmod p$ is uniformly distributed for random $r \in Z_p^*$ and (2) $DLP_{p,g}(a) = DLP_{p,g}(ag^r \bmod p) - r$.

2.2 Indistinguishable Probability Distributions

The indistinguishability of probability distributions, due to Yao [94] and Goldwasser and Micali [60], combines computational assumptions with randomness in a condition that is useful for defining the security of cryptographic primitives. Informally, two distributions are indistinguishable if a guesser cannot tell them apart. This condition will be used in later subsections to define probabilistic encryption, bit commitment, and zero-knowledge proof systems.

Indistinguishability is formalized through the notion of a "distinguisher," which is a probabilistic algorithm that outputs a boolean value when given one or more elements of a distribution as input. If the distinguisher is run twice, on inputs drawn from each distribution, then the distinguisher is "successful" if the two outputs differ.

Let P and Q be probability distributions on k-bit strings. They are *distinguishable* by D if there is a constant $c > 0$ such that

$$\text{prob}(D(x) \neq D(y) | x \leftarrow P, y \leftarrow Q) \geq \frac{1}{2} + \frac{1}{k^c}.$$

[The notation $x \leftarrow P$ indicates that x is drawn from the distribution P.]

For technical reasons, indistinguishability is defined on *families* of distributions. A "family" of distributions is an indexed collection $\{P_k\}$ of probability distributions, where P_k is a distribution on k-bit strings. Two families of distributions $\{P_k\}$ and $\{Q_k\}$ are "computationally indistinguishable" if, for every D which runs in time polynomial in k, there is a constant $K > 0$ such that P_k and Q_k are not distinguishable by D for all $k > K$. For example, the QRA implies that, for n of the proper form, the families of distributions of quadratic residues modulo n and quadratic nonresidues modulo n are computationally indistinguishable.

Indistinguishability can be generalized to apply to "ensembles" of probability distributions. An ensemble is a family of *parametrized collections* of probability distributions, i.e., $\{P_k(z)\}$, where $P_k(z)$ is a distribution on k-bit strings for each choice of $z \in L_k$ (parameter set) and each choice of k. Fixing a different z from each parameter set L_k "collapses" an ensemble into a family of distributions. Two ensembles are said to be computationally indistinguishable if, no matter how each is collapsed, the corresponding families of distributions are computationally indistinguishable.

Notice that there is a natural ensemble $\{M_k(z)\}$ associated with each probabilistic Turing Machine M and language L: $M_k(z)$ is the distribution of outputs of M on input $z \in L$ of length k. This will be useful in the upcoming definition of zero-knowledge proof systems.

2.3 Trapdoor Functions and Encryption Schemes

A function is "one-way" if it is easy to compute and hard to invert. More formally, suppose that $\{f_k\}$ is a family of functions defined on bit strings, where $f_k : \{0,1\}^k \rightarrow \{0,1\}^k$. The family $\{f_k\}$ is one-way if two requirements are satisfied: (a) there is a deterministic polynomial-time (in k) algorithm to compute each f_k; and (b) for every probabilistic polynomial-time algorithm A and for every c there exists a k_c such that

$$\text{prob}(f_k(A(f_k(x))) = f_k(x) | x \leftarrow \{0,1\}^k) < k^{-c}$$

for all $k > k_c$. The existence of one-way functions implies $P \neq NP$, so it should not be surprising that no definitive examples of such functions have been found; the discrete log functions from Section 2.1.1 is an example of a candidate that is widely believed to be one-way.

A family of one-way functions is "trapdoor" if there exists some secret information (called the "trapdoor information" or "trapdoor key") for each function with which the inverse can be computed (with high probability) in probabilistic polynomial time. More formally, the trapdoor function E and the inverting function D can be considered to be the output of a randomized algorithm K that takes as input a security parameter k. The probability that $D(E(x)) \neq x$ should

be negligible, and, for any p.p.t. algorithm A, the probability that $A(E(x)) = x$ should be less than 2^{-k} (where the probability ranges over the choice of x).

A trapdoor permutation (i.e., a trapdoor function that is a bijection) can be the basis for a public-key encryption scheme. The permutation is made public, while the trapdoor key is kept secret. Anyone can encrypt a message by applying the permutation to it, but only the holder of the trapdoor key can invert the permutation to recover the message. No permutation is known to be trapdoor, but one candidate is the RSA encryption function [87]: $E(m) = m^e \bmod n$, where $n = pq$, and the trapdoor key is (p, q) (easily samplable given the family parameter $k = |p| = |q|$). The inverse permutation is $D(m) = m^d \bmod n$, where $ed = 1 \bmod (p-1)(q-1)$, which can be efficiently found from the trapdoor key.

Notice that repeated public-key encryption of the same message always yields the same result, i.e., some information about a message is always revealed to an eavesdropper. This leakage is eliminated by using Goldwasser and Micali's "probabilistic public-key encryption" [60], of which the following is an example. If n is a product of two primes each congruent to 3 modulo 4, and if the Quadratic Residuosity Assumption holds, then the inverse of $QRP_n(\cdot)$ is a one-to-many trapdoor function, where the factorization of n is the trapdoor key. Anyone can send a bit stream as a sequence of squares (encrypted 0's) and nonsquares (encrypted 1's) modulo n.

What makes this a probabilistic encryption scheme is that there are exponentially many ways to encrypt each bit (as a function of the length of the trapdoor key), and that these possible encryptions are easy to generate at random. If n is the product of two primes that are congruent to 3 modulo 4, and a is a random integer, then a^2 is a random quadratic residue (square) mod n, and $-a^2$ is a random quadratic nonresidue (nonsquare) mod n. The QRA can be shown to imply the computational indistinguishability of the distribution of encrypted zeros and the distribution of encrypted ones, which in turn implies the computational indistinguishability of the distributions of encryptions of any two messages. Thus no partial information about the message is available (under the QRA) to a probabilistic polynomial-time eavesdropper.

2.4 Bit Commitment

There are two nice properties of a sealable opaque envelope. One is that its contents cannot be changed once the envelope has been sealed (unalterability). The second is that, once sealed, its contents cannot be seen until it is opened (unreadability). A bit commitment scheme is a tool that simulates at least these two properties of opaque envelopes.

A bit commitment scheme can be considered to be a mapping from some large domain to $\{0, 1\}$; a bit is committed by giving a random element in the pre-image of the mapping at that output value. The scheme is unalterable if the mapping is in fact a function. The scheme is unreadable if the distributions of elements in the pre-image of zero and elements in the pre-image of one are indistinguishable to the receiver.

The first bit commitment protocol was part of a protocol by Blum [24] for two-party coin flipping. A commitment by one party, followed by a guess by the second party, followed by a revelation by the first party, is equivalent to the flip of a coin. Naor [79] shows a general construction for basing bit commitment on any one-way function, based on earlier reductions [66] [63].

A more specific example of a bit commitment scheme is based on quadratic residuosity. If $n = pq$, $p = q = 3 \bmod 4$, then a bit is committed by sending a quadratic residue modulo n (for a zero) or a quadratic nonresidue modulo n (for a one). The scheme is unalterable, since no element

can be both a residue and a nonresidue; the scheme is unreadable by a polynomially bounded receiver under the QRA.

Basic bit commitment requires that the receiver be polynomially-bounded. Another flavor of bit commitment, called "strong bit commitment" [28] [30] [80] allows the receiver to be unbounded. In this case, unreadability requires that the two probability distributions be (almost) identical.

2.5 Interactive Proof Systems and Zero-Knowledge Proof

An interactive proof system is a two-party protocol in which a "prover" conveys a convincing argument to a polynomially-bounded probabilistic "verifier;" this idea was introduced by Goldwasser, Micali, and Rackoff [61] and Babai [3]. More precisely, given a language L, an interactive proof system for L consists of two probabilistic interactive machines called the prover and the verifier that have access to a common input tape containing a possible element x of L, and that send messages through a pair of communication tapes, but otherwise are inaccessible to one another. The verifier performs a number of steps polynomial in the length of the common input, ending in either an "accept" or a "reject" state. When the verifier behaves correctly, then (1) when $x \in L$, the probability that the verifier rejects is less than any reciprocal polynomial in the length of x; and (2) when $x \notin L$, the probability that the verifier accepts is also less than any reciprocal polynomial in the length of x, even when the prover behaves in an arbitrarily adversarial manner.

An interactive proof system for a language L is "computational zero-knowledge" [61] if the verifier learns nothing from the interaction except the validity of the proof, even if the verifier behaves in an arbitrarily adversarial (probabilistic polynomial-time) manner. This vague condition can be formalized in terms of the computational indistinguishability of ensembles of probability distributions. Consider the probability distribution generated by an adversarial verifier, called its "view," which covers everything seen by the verifier during an interactive proof: messages from and to the prover, plus verifier's coin tosses. For each adversarial verifier, there is an ensemble $View_k(z)$ of such distributions, parameterized over all $z \in L$ of length k. A proof system for membership in L is zero-knowledge if the view ensemble of any adversary V^* is computationally indistinguishable from the natural ensemble associated with L and some probabilistic polynomial time Turing Machine M_{V^*} (called the "simulator" for V^*).

A common technique for the zero-knowledge proof of a proposition is a "cut-and-choose" procedure. The proposition is "scrambled" in some way and presented to the verifier. The verifier challenges the prover to either (a) prove the scrambled proposition, or (b) prove that the scrambled proposition is true if and only if the original proposition is true. Either (a) or (b) alone should reveal nothing to the verifier about the proof of the original proposition; cheating by the prover is caught at least half the time. By iterating this procedure, exponentially high confidence can be achieved by the verifier after a linear number of challenges.

As an example of a cut-and-choose procedure, consider the following sketch of a zero-knowledge proof system for Hamiltonicity [26]. A Hamiltonian cycle in a graph is a set of edges that connects all vertices in a simple (non-self-intersecting) loop. Suppose that the prover wishes to convince the verifier that a graph G with n nodes has a Hamiltonian cycle. The prover takes the adjacency matrix for G (an n by n binary matrix whose (i, j) entry is 1 if there is an edge from node i to node j and zero otherwise), randomly permutes the rows, randomly permutes the columns, and replaces each bit with a commitment for that bit. This scrambled adjacency matrix is sent to the verifier, who responds with one of two challenges: either (a) require the prover to open only those

n bit commitments that correspond to the Hamiltonian cycle, or (b) require the prover to open all bit commitments and properly reorder the rows and columns.

The main importance of zero-knowledge proof systems to secure distributed computation arises from the fact that every language in NP has a zero-knowledge proof system under the assumption that a one-way function exists (first shown by Goldreich, Micali and Wigderson [56] under the assumption that trapdoor permutations exist). In particular, the set of correct messages that can be sent by any player at any moment under any protocol is a language in NP. Thus every message that is sent in a protocol can be accompanied by a zero-knowledge proof that it is an appropriate message. In this way, the encrypted messages of any cryptographic protocol can be "validated via zero-knowledge proof," which can protect against faulty players that might try to violate the protocol. The use of validation in secure distributed computation protocol will be seen in Sections 2.4.3 and 2.5.3. We also mention zero-knowledge proof of knowledge [51] [91], in which the verifier is convinced that the prover holds an undisclosed witness for language membership; these proof systems are also important in the context of protocol validation.

2.6 Secret Sharing and Verifiable Secret Sharing

Underlying many recent results in secure distributed computing is a secret sharing scheme due to Shamir [88]. In this scheme, a secret s is shared by a "dealer" among n players by giving each player a point on an otherwise random degree t polynomial $p(x)$ such that $p(0) = s$ (e.g., give $p(i)$ to the ith player). Any $t+1$ players can interpolate to recover $p(x)$ and hence s. If all computation is performed over a finite field, then the secret is secure from any group of up to t players in the strongest possible (information-theoretic) sense. The parameter t is called the "threshold" of the secret sharing scheme.

When a secret s is shared among n players using a scheme with threshold t, then the piece given by the dealer to a single player may be called a "t-share" of s. Notice that this protocol has two phases. The dealer distributes t-shares during a "sharing phase," and the players interpolate to recover the secret during a "recovery phase."

Verifiable secret sharing (VSS), first introduced by Chor, Goldwasser, Micali, and Awerbuch [38], is a form of secret sharing in which cheating by the dealer and some of the players cannot prevent the honest players from receiving valid shares of some unique recoverable secret. More formally, a t-VSS scheme has three properties: (1) If the dealer is honest, then the secret is information-theoretically secure during the sharing phase from any group of up to t players; (2) The value that will be recovered by the honest players during the recovery phase is uniquely determined at the end of the sharing phase; and (3) If the dealer is honest during the sharing phase, then the recovered value will be the dealer's actual secret. This primitive is also used extensively in the secure computation protocols given in this paper.

To illustrate the idea, we describe a VSS scheme due to Ben-Or, Goldwasser, and Wigderson [20] that can correct up to t player errors, with no probability of errors, if $n > 3t$. The dealer chooses a degree t polynomial $p(x, y)$ in *two* variables x and y, such that the secret $s = p(0, 0)$. Instead of sending single values to other players, each player i receives from the dealer two degree t polynomials in *one* variable, derived from $p(x, y)$: e.g., player i receives primary shares $f_i(y) = p(i, y)$ and $g_i(x) = p(x, i)$.

Upon receiving two univariate polynomials $f_i(y)$ and $g_i(x)$, each player i does a secondary share with each player j: e.g., i sends to j the single secondary share value $s_{ij} = f_i(j)$. Notice that if

both the dealer and player i are honest then $s_{ij} = g_j(i)$ as well.

Dishonesty by either the dealer or other players will cause inconsistencies among the primary and secondary shares received by the honest players. For each inconsistency, a challenge is made to the dealer to send to all players some relevant share information. If the inconsistencies convince a player that the dealer is cheating, then the challenge is accompanied by an "accusation" of the dealer. For example, if $f_i(i) \neq g_i(i)$, then player i accuses the dealer of cheating, and challenges the dealer to send $f_i(x)$ and $g_i(x)$ to all players. Similarly, if $g_i(j) \neq s_{ji}$ for more than t j's, then i accuses the dealer of cheating, and challenges the dealer to reveal i's primary shares. If $g_i(j) \neq s_{ji}$ for between 1 and t j's, then the dealer is not accused (the secondary sharers may be cheating); however, the dealer is challenged by i to send the suspicious secondary shares (s_{ji}'s) to all players.

As the dealer responds to challenges, more inconsistencies will be revealed, causing more accusations and challenges to be made. After a constant number of iterations, either the dealer is repudiated (more than t players have accused the dealer of cheating, or the dealer is upheld (at most t accusations have been made). If the dealer is repudiated, then some default value is chosen for its secret (e.g., $s = 0$), and some default share value is chosen by each player (e.g., $s_i = i$). If the dealer is upheld then i takes its share to be $f_i(0)$, where $f_i(x)$ is either from the original primary share (if i never accused the dealer of cheating) or from the revealed primary share (if the dealer responded to an accusation by i of cheating).

In later sections, this VSS scheme will be used in secure computation protocols, as will a VSS scheme due to T. Rabin [85] that can tolerate any minority of faulty players with an exponentially small probability of error.

2.7 Instance Hiding

Instance hiding schemes ([1], [11]) are a means for computing with encrypted data. Assume that a computationally weak player wishes to compute a complicated function with the help of powerful but untrusted players (called "oracles"). The idea is for the weak player to ask questions of the powerful players from which the desired answer can be easily computed, but in such a way that sufficiently small subsets of the powerful players cannot determine anything about your intended computation from those questions. Every function has a multiple oracle instance hiding scheme that leaks no information to any single oracle [11] (but which allows any two oracles to determine the intended computation).

As an example, here is a 1-oracle instance-hiding scheme for computing the discrete logarithm. To find $DLP_{p,g}(a)$, choose a random i, $0 < i < p$, and ask the oracle for $DLP_{p,g}(ag^i)$. The oracle will reply with some j such that $g^j = ag^i \bmod p$, and thus $j - i$ is the desired result. The oracle learns nothing about the value of a for which the answer was originally sought, since the distribution of ag^i is uniform for any a if i is chosen uniformly. Notice that this instance-hiding scheme relies on the random self-reducibility of the discrete logarithm function.

Instance hiding may be thought of as a close cousin to secure distributed computation. It is listed in this section on preliminaries because two interesting protocols for secure distributed computation are in fact based directly on instance hiding as a primitive [2] [12].

3 Models and Basic Protocols

In this section, the background definitions and concepts are presented for secure distributed computing. Two fundamental protocols are formally described: Oblivious Transfer [?] and Byzantine Agreement [74]. Some of the difficulties of formalizing security for protocols are discussed, along with some recent attempts to overcome these difficulties.

3.1 Protocol Definitions

In this subsection, definitions are given for the three fundamental components of a computation protocol: the parties that participate in the computation, the communication media over which messages are sent, and the adversaries who attempt to defeat the security of the protocol.

Unless otherwise stated, all computation is assumed to be over some large finite field.

Parties: Each party (or "player" or "processor") in a protocol is modeled as an "interacting Turing machine." Each such machine has a private input tape (read-only), output tape (write-only), random tape (read-only), and work tape (read-write). All of the machines share a common input tape (read-only) and output tape (write-only). In addition, each machine may have a history tape, on which it may record any computation or commmunication in which it is involved. Other tapes may be associated with machines or with pairs of machines to model the communication media available for the protocol; this will be covered shortly.

For most of the protocols discussed in this paper, either $n = 2$ ("two-party") or $n \geq 3$ ("multi-party"). The ith processor, $1 \leq i \leq n$, has the secret value s_i (usually an element of a finite field) on its private input tape. A description of the function $f(x_1, \cdots, x_n)$ (e.g., the encoding of a circuit that evaluates the function) is on the common input tape. A protocol specifies a program for each processor, at the end of which each processor has the value $f(s_1, \cdots, s_n)$ on its output tape. Assume that the programs of the processors coordinate all messages to be sent and received at the ends of "rounds" (i.e., at the ticks of a global clock, between which all local computation of all processors is done asynchronously).

If the protocol is "cryptographic," i.e., if it relies on some unproven intractability assumption, then the program for each interactive Turing Machine is assumed to run in probabilistic polynomial-time. Otherwise, the protocol is "non-cryptographic" (or "unconditional"), and there are no bounds on the running time of the programs. However, for most non-cryptographic protocols described in this paper, the programs for all players are polynomial-time (two exceptions are Bar-Ilan and Beaver [5] and Beaver, Feigenbaum, Kilian, and Rogaway [12]).

Some minor variations in the definition of protocol appear in the literature. Two of these variations are mentioned at this time.

The definition of protocol assumes that each player receives the result of a single computation. One variation is to assume that there is a separate function for each player, and that the protocol ends with the output of each function written on the corresponding player's output tape. This variation can be seen to be nearly equivalent to the definition presented. Simply augment each player's private input by adding a private random value, and have the common output be a tuple of private output values added to private random values. Then everyone receives all outputs, but only one of those outputs is recoverable by each player. However, this transformed single-output version may be more vulnerable than the original to a different kind of attack. If a collection of players halt the protocol prematurely, they may gain more information about the outputs than the

remaining players. Protocols that can withstand this sort of attack are called "fair;" fairness is discussed further in Section 2.4.4.

For two-party protocols, the definition assumes that both players know the function, and each player has a private input. It is equivalent to say that only one player knows the function. Simply represent the function as a circuit, encode the circuit as a description, and have that circuit description be an additional input to a universal circuit. This interchangeability of data and function is a general phenomenon that can be exploited in protocols for secure computation.

Communication Models: In general, players communicate with one another through private channels or public channels (or both). A message sent through a private channel is inaccessible to anyone except the designated sender and receiver. A public channel, called a "broadcast" channel, sends a message to everyone. By labeling a message on a broadcast channel, it may be addressed to a single receiver, but its contents are accessible to all players.

If a pair of players are connected by a private channel, then the corresponding pair of interactive Turing Machines share two additional "communication" tapes. Each tape is exclusive-write for one of the two players (unerasable, with a unidirectional writing head), and is readable by both players. If there is a broadcast channel, then each interactive Turing Machine has an exclusive-write tape that is readable by all other players.

The cryptographic protocols typically assume only a broadcast channel (in addition to whatever cryptographic assumptions are needed). The non-cryptographic protocols typically assume a complete (fully connected) network of private channels among the players. Several of the non-cryptographic protocols assume a broadcast channel as well. Some cryptographic protocols assume only a complete network of "oblivious transfer channels" (i.e., private channels that allow OT by some means), and in fact require no additional cryptographic assumptions. Some work has been done in the unconditional setting with incomplete networks of private channels [86] [48].

Adversaries: A player is considered "correct" if it follows its program exactly, engaging in no additional communication or computation beyond what is specified by the protocol, and keeping all of its private tapes private. A player that is not correct is considered "faulty." To model worst-case behavior (maximally malicious coordination), the faulty players in a protocol are modeled as being under the control of a single adversary. In the cryptographic setting, the adversary is a probabilistic polynomial-time algorithm. In the unconditional setting, the adversary is an algorithm with no time or space bounds.

A "passive" adversary can read the private internal tapes and received messages of all faulty processors, but does not interfere with the programs of the faulty processors. This models the situation in which a group of dishonest players participate in the protocol properly, and afterwards pool their information in an attempt to learn more about the inputs of the honest players.

An "active" adversary can read private tapes and messages of faulty players, and can also specify the messages that are sent by faulty players. In other words, an active adversary can cause the faulty processors to violate the protocol in an arbitrary coordinated attack. An active adversary is "rushing" if it can read all messages sent to the faulty players in a round before deciding on the messages sent by those faulty players in that same round.

An adversary can also be either "static" or "dynamic." A static adversary controls the same set of faulty players throughout the protocol. A dynamic adversary can increase the set of faulty players under its control each round. Even within a single round, faulty players may be added dynamically, on the basis of messages and internal tapes of those players that have already been

corrupted.

Other types of adversaries are seen less frequently in the secure distributed computing literature. Fail-stop adversaries [55] are active adversaries that are limited to withholding outgoing messages from a faulty processor. Independent adversaries [48] are two or more adversaries (e.g., one or more passive and one or more active) for a single protocol, which control possibly overlapping sets of faulty processors, and which possibly cannot communicate during the course of the protocol.

For most of the protocols discussed in this paper (but not all [83] [48]), there is either a single static passive adversary or a single static active adversary.

3.2 Basic Protocols

This section describes two basic protocols that are useful primitives for secure distributed computation: Oblivious Transfer and Byzantine Agreement.

Oblivious Transfer: Suppose that player A has a secret bit b that is unknown to player B. The two players engage in a protocol, at the end of which player B successfully receives bit b with probability $\frac{1}{2}$. B always knows with certainty whether or not the "transfer" of bit b was successful. By contrast, A cannot determine that the transfer was successful any better than random guessing (or non-negligibly better than random guessing, with respect to some security parameter). This is a description of the simplest version of Oblivious Transfer, due to Rabin [?] (abbreviated "OT"). It is analogous to mail service by an incompetent Postal Service; any sent letter arrives at the proper destination half of the time, and disappears the other half of the time.

The input-output requirements of this simple Oblivious Transfer protocol can be described in the secure computation model given earlier in this section. Player A begins with the private input (b, r), where b is the secret bit, and where r is a private random bit. Player B begins with the private input (m, r'_1, r'_2), where all three components are random bits. At the end of the protocol, each player has the output value $f((b, r), (m, r'_1, r'_2))$ on its output tape, where $f((b, r), (m, r'_1, r'_2)) = (m \oplus b, r'_2)$ if $r = r'_1$, and $= (m, 1 \oplus r'_2)$ if $r \neq r'_1$.

There are many other types of Oblivious Transfer that have all been shown to be equivalent to the simple one [29] [46] [47]. One important alternative is called "1-2 Oblivious Transfer" [50] (abbreviated "1-2-OT"). In this version, player A begins with two secret bits b_0 and b_1. Player B can choose to receive exactly one of these bits, without letting A know which bit was chosen.

Oblivious Transfer is one of the most basic possible primitives that can break the "knowledge symmetry" between two players. In addition to other versions of Oblivious Transfer, more sophisticated protocols can be built from this primitive. In fact, secure distributed computation can be reduced to Oblivious Transfer [70] [13] [59], as will be discussed in later sections. To give a simpler reduction now, the following scheme, due to Crépeau [46], achieves bit commitment through oblivious transfer:

1. Player A chooses random $b_1, \cdots, b_n$ such that $b_1 \oplus \cdots \oplus b_n = b$.

2. Player A sends each b_i, in order, to player B by oblivious transfer. At this point, player A has committed to bit b.

To reveal the committed bit b, player A sends each b_i, in order, to player B without using oblivious transfer. Player B rejects the commitment if any b_i disagrees with a b_i that was successfully

received earlier, and accepts $b = b_1 \oplus \cdots \oplus b_n$ otherwise. The probability of A cheating B is at most $\frac{1}{2}$, and can be reduced to 2^{-k} by parallel k-wise execution of the basic protocol.

These Oblivious Transfer primitives, and especially 1-2-OT, will be used frequently in the computation protocols discussed in this paper.

Byzantine Agreement: Suppose that there are n players, each player i has a single bit b_i, and some of the players are unreliable. The Byzantine Agreement problem is to devise a protocol at the end of which all of the non-faulty players agree on a bit b (called the "consensus value"). Moreover, we must have that $b = b_i$ if i is honest and all of the honest players had the same initial bit. This problem was initially considered by Lamport, Shostak, and Pease [74], who showed that it can be solved if and only if less than one-third of the players are faulty. As will be shown in the next subsection, Byzantine Agreement can be used to eliminate the need for a broadcast channel in some secure distributed computation protocols.

This protocol can also be described using the model for fault-tolerant secure computation (although the purpose in not to conceal the inputs, and although channels are not private in the original model[1] [74]). Suppose that the consensus value will be 0 when the honest players do not have the same initial bit. Then each player i begins with private input b_i on its tape. At the end of the protocol, each player has the value $f(b_1, \cdots, b_n)$ on its output tape, where $f(b_1, \cdots, b_n) = b_1$ if $b_1 = \cdots = b_n$ and $= 0$ otherwise. The requirement is that this protocol remain correct in the presence of an active adversary; a more precise condition is to say that the protocol is "resilient," which will be defined in the next subsection.

3.3 Security Definitions

There are two important security properties of computation protocols that are considered in this paper: "privacy" and "resilience." These terms capture the ability of a protocol to withstand some fraction of faulty players that are under the control of an adversary.

Privacy assumes a passive adversary. A protocol is said to be "t-private" if, given any set of at most t faulty processors, no passive adversary can learn anything more about the private inputs of the honest players than is otherwise revealed by knowing the output of the computation and knowing the private inputs of the faulty players. This can be formalized by considering whether there is an algorithm, of the same computational power as the adversary, that, given only the output and faulty players' inputs, can in some sense simulate anything achievable by the adversary throughout and at the end of the protocol.

More precisely, we can say that a given coalition C learns no useful information when performing a protocol to compute a function f, if the following holds: For every two input vectors $\vec{s}, \vec{s}'$ such that $f(\vec{s}) = f(\vec{s}')$ and such that $\vec{s}_C = \vec{s}'_C$, the distributions of messages exchanged between C and $\bar{C}$ are computationally indistinguishable, where the distributions are over the random tapes of $\bar{C}$.

Resilience assumes an active adversary. A protocol is said to be "t-resilient" if no active adversary, with at most t faulty processors under its control, can (1) learn anything more about the private inputs of the honest players than is otherwise revealed by knowing the output of the computation and knowing the private inputs of the faulty players; or (2) prevent any honest player from having the correct result on its output tape at the end of the protocol.

[1] A fast probabilistic protocol for Byzantine Agreement in a network of private channels is due to Feldman and Micali [52]; this is closer to the model that we consider in this paper.

Note that the notion of "correct result" in the definition of t-resiliency is not straightforward. Certainly the adversary could have one or more faulty players behave as though they held a different input than they actually did, while otherwise being in perfect accordance with the protocol. This behavior cannot be protected against by the honest players. What can be said is that if each player initially commits to an input (e.g., using a verifiable secret sharing scheme) then the honest players must learn the value of the function at those committed inputs; cheating players who fail to commit are eliminated from the computation. In addition to maintaining privacy, initial commitment also guarantees "independence of inputs," i.e., the faulty players' inputs cannot be related in any way to the honest players' inputs. For example, the margin of victory of a secret-ballot election could be influenced by announcing votes as they were made.[2]

Notice also that under certain conditions a protocol that requires a broadcast channel is equivalent to a protocol that requires only a network of private channels. Obviously, if the adversary is passive, then each broadcast can be simulated by sending the same message privately to each player. More interestingly, if the adversary is active, then each broadcast can also be simulated on a private network if there are sufficiently few faulty players. Using Byzantine Agreement [74] [52], a t-resilient protocol that requires a broadcast channel can be converted into a t-resilient protocol that needs only a network of private channels, whenever $n > 3t$.

3.4 Unifying Protocol Models

Rather than proving the above properties directly and individually, one may try to somehow capture all of the "good" properties of a secure protocol in one crisp definition. This is an important endeavor, since it can lead to formal proofs of properties we would require of general protocols (although that level of formality is beyond the scope of our overview). In this subsection, two proposals for such a definition are described. A third proposed definition, due to Goldwasser and Levin [59], will not be discussed.

One proposal to put this area on a more formal foundation is due to Beaver [9]. He defines the idea of an "interface" to allow an adversary for one protocol to attack a second protocol; an interface is a Turing Machine that sends messages to the adversary and sends how-to-corrupt instructions to the second protocol. Two protocols can then be defined to be equally resilient if there is an interface such that the output distribution of the first protocol with any adversary is indistinguishable from the output distribution of the second protocol with the same adversary through an interface. The output distribution covers both the information that is available to the adversary at the end of the protocol and the influence that the adversary can have on the final state of the players of the protocol. A similar definition of equally private protocols can be given.

Given these notions of relative resilience (privacy), a protocol can be defined to be resilient (private) if it and an "ideal" protocol are equally resilient (private). Here an ideal protocol is a protocol that can make use of additional trusted parties that no adversary can influence. Among the strengths of this definition is that it provides a single unifying security measure, and that it facilitates modularity in both protocol design and proof of security.

Another proposed formalization, due to Micali and Rogaway [78], builds on the successful definition of the zero-knowledge property for interactive proof systems. They define a "simulator" for a protocol, which can interact with an adversary in place of the network of processors. In

[2] This is one reason why exit poll data is not released until after all polls close nationwide on Election Day.

addition to reading all private tapes of the adversary at all times, the simulator can access certain limited information about the private inputs by querying a certain oracle. The oracle only allows queries about private inputs and computed outputs that are consistent with the power of the adversary. For example, if an active adversary can dynamically corrupt up to t players, then t input queries and one output query can be made by the simulator. The single output query must be made at inputs that agree with the real private inputs everywhere except possibly on those components about which input queries have been made. Furthermore, that query must use values for the queried components that are a function only of the adversary's current history (e.g., not a function of the simulator's coin flips).

There are two ensembles of distributions of interest. One is the view of an adversary attacking the protocol, which includes the tape contents of and messages to all corrupted players. The second is the simulated view, which includes only the output of the simulator after interacting with the adversary. A protocol is resilient if, for any adversary, there exists a simulator such that these two ensembles are indistinguishable.

4 Two-Party Cryptographic Secure Distributed Computation

4.1 Main Ideas

In this subsection, we present two-party computation protocols whose security rely on cryptographic assumptions. There are two types of protocols, both of which work from the circuit representation of the computed function. The first type has two largely non-interactive subprotocols; one player "scrambles" the circuit in some manner, then they interact, and then the second player "evaluates" the scrambled circuit. The second type is an interactive gate-by-gate evaluation of the circuit for encrypted inputs.

Note that secure two-party computation is not possible in general without some complexity assumption. As a simple example, Ben-Or, Goldwasser, and Wigderson [20] show that there cannot be an information-theoretically secure two-party protocol for computing the OR of two input bits. There are other impossibility results for unconditional two-party computation that are covered in Section 2.6.

The first results in two-party secure computation are due to Yao [94]. He was interested in a setting in which both parties behaved honestly, but wished to maintain the secrecy of their inputs ("1-privacy" as defined in Section 2.2.3). One of the problems he considered was called the "Millionaires' Problem." Two millionaires wish to know who is richer, without revealing any other information about each other's net worth. One of the solutions proposed for this problem depends on the existence of a public-key cryptosystem; two others were mentioned that relied on commutative one-way functions and probabilistic encryption respectively. Other applications mentioned included secret-ballot elections and private database queries. In this same paper, Yao indicated without details that any 2-ary function $f(x, y)$ could be computed privately by two players assuming the hardness of factoring.

4.2 Protocols

The first general two-party secure computation protocol result is due to Yao [95], who considers a task that is somewhat more general than the computation problem described in Section 2.2. Instead

of computing a deterministic function, an "interactive computational problem" has inputs and outputs that are distributed according to given probability distributions. This reduces to function evaluation when, for each pair of inputs, there is only one element of the output distribution with non-zero probability. Assuming the intractability of factoring, Yao shows that every two-party interactive computational problem has a private protocol.

Goldreich, Micali, and Wigderson [57] show how to weaken the intractability assumption from factoring to the existence of any trapdoor permutation. In fact, their protocol solves the following slightly different, but equivalent, problem of "combined oblivious transfer." There are two parties A and B, with private input bits a and b respectively. At the end of the protocol, A is to receive the value $f(a,b)$ without learning anything further about B's input bit b. Meanwhile, B is to learn nothing at all about A's input bit a.

We sketch this protocol for combined oblivious transfer. Assume that $E(M,r)$ is a probabilistic encryption function for the message M and the random key r, such that M is easily recovered from $E(M,r)$ and r. Suppose that the circuit C for the function $f(x,y)$ consists of gates $g_1, \cdots, g_m$, where each g_i is either a binary AND gate or a unary NOT gate. Assume that the final output of the circuit is the output of the gate g_m. For simplicity of notation, consider the inputs x and y to be 0-ary gates g_x and g_y. Let the encryption function E take as input an s-bit message string m and an s-bit random string r (for some security parameter s). As usual, "$\oplus$" denotes either the exclusive-or of two bits or the bitwise exclusive-or of two binary strings.

The players proceed as follows, with a "circuit construction" phase by B and then a "circuit evaluation" phase by A:

1. B chooses random s-bit strings $r_1, r_1', \cdots, r_m, r_m', r_x, r_x', r_y, r_y'$.
2. B chooses random bijections $\phi_1, \cdots, \phi_m, \phi_x, \phi_y$ such that ϕ_i maps $\{r_i, r_i'\}$ to $\{0,1\}$.
3. For each AND gate g_i, B constructs the set S_i as follows:
   (a) Find the gate g_{left} whose output feeds into the left input of g_i.
   (b) Find the gate g_{right} whose output feeds into the right input of g_i.
   (c) Choose random s-bit messages $m_{i1}, m_{i2}, m_{i3}, m_{i4}$.
   (d) Compute the related values $m_{i1}', \cdots, m_{i4}'$ as follows:
      i. $m_{i1}' = m_{i1} \oplus \phi_i^{-1}(\phi_{left}(r_{left}) \wedge \phi_{right}(r_{right}))$
      ii. $m_{i2}' = m_{i2} \oplus \phi_i^{-1}(\phi_{left}(r_{left}') \wedge \phi_{right}(r_{right}))$
      iii. $m_{i3}' = m_{i3} \oplus \phi_i^{-1}(\phi_{left}(r_{left}) \wedge \phi_{right}(r_{right}'))$
      iv. $m_{i4}' = m_{i4} \oplus \phi_i^{-1}(\phi_{left}(r_{left}') \wedge \phi_{right}(r_{right}'))$
   (e) Construct the pairs p_1, p_2, p_3, p_4 as follows:
      i. $p_1 =< E(m_{i1}, r_{left}), E(m_{i1}', r_{right}) >$
      ii. $p_2 =< E(m_{i2}, r_{left}'), E(m_{i2}', r_{right}) >$
      iii. $p_3 =< E(m_{i3}, r_{left}), E(m_{i3}', r_{right}') >$
      iv. $p_4 =< E(m_{i4}, r_{left}'), E(m_{i4}', r_{right}') >$
   (f) Let S_i be some random permutation of the set $\{p_1, p_2, p_3, p_4\}$.

4. For each NOT gate g_i, construct the set S_i as follows:

   (a) Find the gate g_{only} whose output feeds into the input of g_i.

   (b) Let S_i be some random permutation of the set $\{E(\phi_i^{-1}(1 \oplus \phi_{only}(r_{only})), r_{only}), E(\phi_i^{-1}(1 \oplus \phi_{only}(r'_{only})), r'_{only})\}$.

5. $B \rightarrow A$: $S_1, S_2, \cdots, S_m, \phi_y^{-1}(b), \phi_m = \{< r_m, \phi_m(r_m) >, < r'_m, \phi_m(r'_m) >\}$.

6. $B \rightarrow A$: $\phi_x^{-1}(a)$ via 1-2-OT (i.e., B offers a choice of $\phi_x^{-1}(0)$ and $\phi_x^{-1}(1)$, and A chooses to receive the first if $a = 0$ and the second if $a = 1$.

7. Player A now proceeds to evaluate the "circuit," as given by the received sets $S_1, \cdots, S_m$. Originally label all of the AND and NOT gates $g_1, g_2, \cdots, g_m$ as "unmarked." At the start of this phase, A is said to "know" the values $\phi_x^{-1}(a)$ and $\phi_y^{-1}(b)$ received in steps 5 and 6. While there remains an unmarked gate, do the following:

   (a) If there is an unmarked NOT gate g_i such that its input wire connects to the output wire of gate g_{only}, and such that player A "knows" one of the elements in the domain of ϕ_{only}, then A can decrypt exactly one of the elements of S_i. Specifically, A can find $D(E(\phi_i^{-1}(1 \oplus \phi_{only}(r_{only})), r_{only}), r_{only}) = \phi_i^{-1}(1 \oplus \phi_{only}(r_{only}))$ if A "knows" r_{only} in the domain of ϕ_{only}; and A can similarly find $\phi_i^{-1}(1 \oplus \phi_{only}(r'_{only}))$ if A "knows" r'_{only}. This decrypted value, an element in the domain of ϕ_i, is now said to be "known" by A. Mark the gate g_i.

   (b) If there is an unmarked AND gate g_i such that its left input wire connects to the output wire of gate g_{left} and its right input wire connects to the output wire of gate g_{right}, and such that player A knows one of the elements in the domain of ϕ_{left} and one of the elements in the domain of ϕ_{right}, then A can decrypt both elements of exactly one pair of S_i. For example, if A "knows" r_{left} in the domain of ϕ_{left} and r'_{right} in the domain of ϕ_{right}, then A can compute $D(E(m_{i3}, r_{left}), r_{left}) = m_{i3}$ and $D(E(m'_{i3}, r'_{right}), r'_{right}) = m'_{i3}$. The bitwise exclusive-or of the decrypted values is one of the elements in the domain of ϕ_i, e.g., $m_{i3} \oplus m'_{i3} = \phi_i^{-1}(\phi_{left}(r_{left}) \wedge \phi_{right}(r'_{right}))$. This element in the domain of ϕ_i is now said to be "known" by A. Mark the gate g_i.

8. At this point, A "knows" one of the elements r in the domain $\{r_m, r'_m\}$ of ϕ_m, where g_m is the final output gate. The final output $C(a, b)$ is taken to be $\phi_m(r)$.

It is straightforward to show that, if B performs the construction correctly, then A can mark all of the gates. Furthermore, the final value $\phi_m(r)$ can be shown to be the correct value of the circuit $C(a, b)$, as follows. Each occurence of step 7a maintains the following invariant property: If A "knows" the element r^*_{only} at the start, and "knows" the element r^*_i at the end, then $\phi_i(r^*_i) = 1 \oplus \phi_{only}(r^*_{only})$. Each occurence of step 7b maintains the following invariant property: If A "knows" the elements r^*_{left} and r^*_{right} at the start, and "knows" the element r^*_i at the end, then $\phi_{left}(r^*_{left}) \wedge \phi_{right}(r^*_{right}) = \phi_i(r^*_i)$. Thus, in both cases, $\phi_i(r^*_i)$ is the value computed by the gate g_i. At the end A knows r^*_m, and, since A received ϕ_m in step 5, A can compute $\phi_m(r^*_m) = C(a, b)$, the value of the output of C.

It can also be shown that A learns no additional useful information about B's input. For example, in step 7b, A can decrypt single elements from some of the pairs $p_1, \cdots, p_4$, but can only decrypt both values from a single pair. Since the m_{ij} values are random, a single element from a pair yields no useful information.

Note that trapdoor permutations are needed in this protocol to implement 1-2-OT (via a construction due to Even, Goldreich, and Lempel [50]).

Galil, Haber, and Yung [55] show how to reduce further the complexity assumption from trapdoor permutation to one-way function plus Oblivious Transfer. As before, one player is the "circuit" constructor, and the other is the "circuit" evaluator. The constructed circuit consists of a number of "gates," each of which enables a single decryption key (output) to be recovered from the knowledge of two decryption keys (inputs). The input decryption keys serve as seeds for a pseudorandom number generator (which can be based on any one-way function [66] [63]) that returns the output decryption key (yet another seed for the generator).

Kilian [70] shows how to base oblivious circuit evaluation solely on Oblivious Transfer as a primitive (black-box reduction). In Section 2.2.2, we showed Crépeau's bit commitment scheme based on Oblivious Transfer [46]. At the heart of Kilian's construction is a method for extending this commitment scheme so that any NP statement can be proven about committed bits in a zero-knowledge fashion (i.e., without revealing anything further about the committed bits). The circuit evaluation protocol takes advantage of the equivalence (shown by Barrington [6]) of NC^1 functions and width 5 permutation branching programs. Instead of scrambling a circuit, the appropriate permutation programs are randomized in a simple and straightforward manner; this leads to a constant-round evaluation protocol when the underlying function is in NC^1. Any polynomial-sized circuit can then be evaluated by considering it to be a cascade of NC^1 subcircuits (and being careful to enforce consistency between inputs and outputs of connected subcircuits).

Unlike the previous protocols, a protocol due to Chaum, Damgård, and Van de Graaf [36] for two-party secure computation has both parties contribute to the scrambling of the circuit. This protocol requires a bit commitment scheme with additional nice properties: "blinding" (i.e., the recipient of a committed bit can compute a commitment of any bit xored with the committed bit), and "comparability" (i.e., the recipient of two committed bits can be convinced by the committer that they are equal). Chaum, Damgård, and Van de Graaf show how to achieve such an enhanced bit commitment scheme based on any one of several number-theoretic problems: Quadratic residue, discrete log, or Jacobi symbol (even though the Jacobi symbol can be efficiently computed). For example, a strong bit commitment scheme with blinding and comparability can be based on the discrete log under the DLA. A bit b is unconditionally hidden as $a^b g^r \bmod p$ for prime p, generator g of $Z_p{}^*$, fixed $a \in Z_p{}^*$, random $r \in Z_p{}^*$ (i.e., even if discrete log is easy, b cannot be guessed from $a^b g^r \bmod p$ without guessing r). It is then conditionally unforgeable: finding a random r' such that $a^{1-b} g^{r'} = a^b g^r \bmod p$ is equivalent to finding the discrete log of a. To blind a commitment x with a known bit b', the recipient computes $a^{b'} g^{r'} x \bmod p$ for random r'. To compare commitments x and y, the committer reveals the discrete log of $a^{-1}xy$ if $x \neq y$, and reveals the discrete log of $x^{-1}y$ if $x = y$.

The discussion of the protocol itself is deferred to the section on multi-party cryptographic computation, since the same general protocol can handle any number of players.

Abadi and Feigenbaum [2] present a two-party computation protocol that is similar to the protocol of Chaum, Damgård, and Van de Graaf. This protocol is described using the idea of

instance hiding from an oracle, which was discussed in Section 2.1.7. The protocol allows one player to keep a circuit hidden unconditionally (except for the number of AND gates), while the other player can hide the input to the circuit conditionally (i.e., under the QRA). Since circuit can become data (i.e., as a description to be input into a universal circuit), and since data can become circuit (i.e., hardwired into a universal circuit), the same protocol can provide unconditional security for data with conditional security for the circuit.

Let $E_k(x)$ denote the encryption of x using the probabilistic encryption scheme based on quadratic residuosity described in Section 2.1.3: $E_k(x)$ is a random square modulo k if x is 0, and a random nonsquare otherwise. D_k is the corresponding decryption function, which is hard (under the QRA) unless the factorization of k is known.

Let A be the circuit hider, and let B be the data encrypter. Suppose that the circuit C, consisting of some number of binary AND gates and unary NOT gates, is known to player A, and that the input data $x_1, x_2, \cdots, x_m$ is known to player B. The players proceed as follows:

1. B chooses $k = pq$, where p and q are secret primes congruent to 3 mod 4.
2. B encrypts each bit x_i as $y_i = E_k(x_i)$.
3. $B \rightarrow A : y_1, y_2, \cdots, y_m$.
4. While there still remains some gate g_i such that A knows the encryption of all inputs but does not know the encryption of the output, do the following:
   (a) If g_i is a NOT gate with encrypted input y, then A finds the encryption of the output to be $-y$ (since $-E_k(b) = E_k(1-b)$ always for k of this form).
   (b) If g_i is an AND gate with left encrypted input y_1 and right encrypted input y_2, then A finds the encryption of the output interactively, as follows:
      i. A chooses random elements r_1, r_2 and random bits b_1, b_2 and computes $y_1' = (-1)^{b_1} {r_1}^2 y_1$ and $y_2' = (-1)^{b_2} {r_2}^2 y_2$.
      ii. $A \rightarrow B : y_1', y_2'$
      iii. B computes $b_1' = D_k(y_1')$ and $b_2' = D_k(y_2')$
      iv. $B \rightarrow A : z_{00} = E_k(b_1' \wedge b_2')$, $z_{01} = E_k(b_1' \wedge (1 \oplus b_2'))$, $z_{10} = E_k((1 \oplus b_1') \wedge b_2')$, $z_{11} = E_k((1 \oplus b_1') \wedge (1 \oplus b_2'))$
      v. A determines the encryption of the output of the gate to be z_{b_1,b_2}.

   At this point, A holds the encryption z of the output of the entire circuit. The protocol can end to give the output exclusively to either player, as follows:

5. To give the output exclusively to player B, do the following:
   (a) A chooses a random r.
   (b) $A \rightarrow B : z' = r^2 z$
   (c) B finds the output to be $D_k(z')$
6. To give the output exclusively to player A, do the following:

(a) A chooses a random r and a random bit b.

(b) $A \rightarrow B : z' = (-1)^b r^2 z$

(c) $B \rightarrow A : c' = D_k(z')$

(d) A finds the output to be $c' \oplus b$.

If both parties follow the protocol, then exactly one of them ends with the output of the circuit. Privacy is maintained due to the QRA and the random self-reducibility property of quadratic residues and nonresidues. Since each AND gate must be evaluated interactively, the data encrypter B learns the number of AND gates in the circuit.

It is interesting to consider the effect of an active adversary on the two protocols described in detail in this subsection. For the protocol of Abadi and Feigenbaum, if the circuit hider is to receive the output, then malicious behavior could be costly. At the final stage, when the output is to be decrypted, the circuit hider could substitute any encrypted bit of interest (multiplied by a random square); for example, the circuit hider could choose to learn one of the input bits instead of the output bit. By contrast, the protocol of Goldreich, Micali, and Wigderson cannot be exploited in this way by an active adversary (unless the 1-2-OT cannot withstand an active attack), since there is virtually no interaction between the two players. Of course, a cheating player in either protocol can cause the other player to receive the incorrect output without detecting that misbehavior has occurred.

5 Multi-Party Cryptographic Secure Computation

5.1 Main Ideas

In the preceding subsection, several protocols were described for securely computing any two-input function to which each of two players held one of the inputs, given some "reasonable" intractability assumption. Does this generalize to more than two players?

One tempting approach would be to reduce an n-input function to a series of 2-input functions, e.g., reduce n-ary addition into the pairwise computation of $n-1$ partial sums. This computes the correct answer, but it risks leaking information to players involved in computing the intermediate results.

In fact, general protocols for conditional (i.e., cryptographic) multi-party computation have been found. Moreover, some of the solutions do rely on reducing the computation to a series of two-party computations, but in a more sophisticated manner.

All of the protocols described in this subsection follow the three-stage paradigm introduced by Goldreich, Micali, and Wigderson [57]: an input sharing stage, a computation stage, and an output reconstruction stage. The idea is that computation is performed on shares of private inputs, ultimately producing shares of the final answer. These shares are combined in the third stage to reveal only the output to all of the players.

Another technique that is used by several of the protocols is to do "second-order" sharing, i.e., sharing of shares. This idea was first exploited by Galil, Haber, and Yung [55], and is useful for allowing players to prove to one another that their actions are consistent and correct, as well as for enhancing fault tolerance.

Channel:	broadcast only
Adversary:	t passive, $t < n$
Security:	trapdoor function
bit complexity:	$O(n^2C)$
round complexity:	$O(n^2D)$.

Table 1: Summary of Goldreich, Micali, and Wigderson 1987 (passive adversary)

Some protocols (beginning with the results of Goldreich, Micali, and Wigderson [57]) that protect against an active adversary use zero-knowledge proofs in the course of the computation. As described in an earlier section, every language in NP has a zero-knowledge proof system. Such a proof system allows anyone to demonstrate membership in the language without leaking any additional information. Consider the set of all legal messages that can be sent at a given moment in the course of a protocol. As this is a language in NP, the actual message sent can be followed by a zero-knowledge proof of membership [56]. This "validates" the message without compromising security. In fact, a zero-knowledge proof of knowledge of membership suffices to validate each message in the protocol.

This subsection is divided into three further subsubsections. The first describes multi-party computation in the presence of a passive ("gossiping only") adversary. The second considers multi-party computation when the adversary is active, and presents new protocols for this problem as well as modifications of passive-adversary protocols. The third gives multi-party computation protocols that achieve "fairness," a condition which limits the advantage that can be gained by the adversary if faulty players quit the protocol before completion.

In this subsection and the next, some of the protocols that are discussed are accompanied by a table summarizing their basic properties. Each table gives the channel assumptions, the type of adversary that can be tolerated, the security conditions, the probability of error (when relevant), the bit complexity of communication, and the round complexity of communication. In these tables, the number of players in the protocol is denoted by n, the size of a circuit is denoted by C, the depth of a circuit is denoted by D, and $B_Z(k)$ and $R_Z(k)$ denote the bit complexity and round complexity of a message validation via zero-knowledge proof (with security parameter k).

5.2 Multi-Party Cryptographic Protocols versus Passive Adversaries

A two-party protocol for any two-input computation, due to Goldreich, Micali, and Wigderson [57], was presented in Section 2.3.2. That protocol assumed only the existence of any trapdoor function. In the same paper, multi-party computation is reduced to a succession of two-input computations, in such a way that privacy is maintained.

Using a technique due to Barrington [6], the depth D circuit to be computed can be represented as a straight-line program, of length at most 4^D, whose inputs are permutations in S_5 (the 120 possible permutations on five distinct elements). Each player can then privately convert his input into a single corresponding permutation. The computation reduces to finding the composition of a string of private and public permutations.

In the first stage of the protocol, each player shares its permutation with all of the players. A

Channel:	broadcast only
Adversary:	t passive, $t < n$
Security:	one-way function plus Oblivious Transfer
bit complexity:	$O(n^2C)$
round complexity:	$O(D)$

Table 2: Summary of Galil, Haber, and Yung 1987

permutation σ is shared by giving a new permutation σ_i to each player i, where $\sigma_1, \cdots, \sigma_n$ is an otherwise random collection of permutations whose composition is $\sigma_1 \circ \cdots \circ \sigma_n = \sigma$.

In the second stage of the protocol, the straight-line program is performed, from left to right, on the shares of the inputs. As the computation proceeds, each player will hold a share of the partial result obtained thus far.

Composition with a public permutation is immediate. If the straight-line program calls for composing a given permutation σ with a public permutation ϕ to get $\phi \circ \sigma$, then the first player replaces its share σ_1 of σ with the the composition $\phi \circ \sigma_1$.

The composition of two permutations is more difficult. It is not enough for each player to take the composition of its two shares, because of the non-commutativity of composition, i.e., $\sigma \circ \sigma' = (\sigma_1 \circ \cdots \circ \sigma_n) \circ (\sigma'_1 \circ \cdots \circ \sigma'_n) \neq (\sigma_1 \circ \sigma'_1) \circ \cdots \circ (\sigma_n \circ \sigma'_n)$. The composition of two permutations reduces to the composition of $2n$ share permutations, but each player's shares are initially "too far apart."

The nice solution is to perform a series of "swaps" of neighboring share permutations. A "swap" is a two-party protocol that inputs two permutations ρ and τ, and outputs two otherwise random permutations ρ' and τ' such that $\rho \circ \tau = \tau' \circ \rho'$. Since this is a well-defined two-input random function, a two-party secure computation protocol for swapping can be implemented (using the general techniques from Section 2.3). This protocol should give only one of the two outputs to each player: ρ' is given only to the player who input ρ, while τ' is given only to the player who input τ. If the swaps are performed purposefully, then one player's two shares are "closer" together after each swap; each player privately finds the composition of its two shares whenever those shares become adjacent. A total of $O(n^2)$ swaps are necessary to maneuver each player's shares into adjacent locations.

It is shown in the paper by Galil, Haber, and Yung [55] how to reduce the intractability assumption from the existence of trapdoor functions to the existence of one-way functions and a subprotocol for Oblivious Transfer. They also show (in joint work with Micali) how to perform private multi-party computation of boolean functino directly, instead of working with straight-line programs and permutations.

In the first stage, each player shares its input bit by giving each other player a one-bit share. The shares are chosen so that their xor yields the input bit, and so that they are otherwise random. Public-key cryptography is used to transmit shares to players. In the second stage, the function is computed by evaluating its corresponding circuit using only the shares of the private inputs. When the evaluation reaches any intermediate wire, each player has a share of the proper value at that wire (for the given circuit and secret inputs). It suffices to show how to evaluate a NOT gate and

an AND gate.

Negation is immediate. A NOT gate is evaluated by having one player (e.g., the last player) take the complement of its share of the input to the gate.

Conjunction is more difficult. There is no private computation that players can perform on only their own shares of the two inputs to yield a proper share of the output. However, as with the previous protocol, there is a two-player protocol which may be repeatedly performed by pairs of players to help compute the final shares (using the general techniques from Section 2.3). Moreover, this two-player protocol is quite simple. One player inputs bits x and y, the second player inputs bit z, and the second player receives the output bit $f((x,y),z) = x \oplus (y \wedge z)$.

Why does this two-party protocol suffice? Suppose that $a \wedge b$ is to be computed, where each player i holds the xor shares a_i and b_i of a and b. Let $r_{i1}, \cdots r_{in}$ be random bits created by player i. Then, for all $i \neq j$, players i and j perform the two-player protocol twice: once to give player j the value $f((r_{ij}, a_i), b_j) = r_{ij} \oplus (a_i \wedge b_j)$, and once to give player i the value $f((r_{ji}, b_j), a_i) = r_{ji} \oplus (b_j \wedge a_i)$.

Furthermore, note that these $n(n-1)$ two-party protocols can all be performed in parallel, unlike the permutation-swappings of the previous protocol. After performing these $n(n-1)$ two-party protocols, each player can assemble a share of $a \wedge b$ by taking the xor of $2n-1$ values available to it: all $n-1$ of its received outputs, together will all $n-1$ of its own random bits, together with the conjunction $(a_i \wedge b_i)$. Simple (boolean) algebra will verify that this gives each player an xor share of $a \wedge b$.

In each of the two preceding multi-party cryptographic protocols, a simpler two-party protocol was repeated many times: permutation "swapping" for [57] "and-xoring," i.e., $f((x,y),z) = x \oplus (y \wedge z)$ for [55]. In each case, a general protocol for two-party secure computation was then invoked to establish that the complete multi-party protocol was possible.

Goldreich and Vainish [58] observe that general two-party secure computation is unnecessary for these two specific cases. Assuming a protocol for Oblivious Transfer, special-purpose private protocols can achieve the necessary aims directly. For and-xoring, the first player can create two secrets $s_0 = x \oplus (y \wedge 0) = x$ and $s_1 = x \oplus (y \wedge 1) = x \oplus y$. Then the second player can choose the appropriate secret s_z by 1-2-OT. For permutation swapping, the first player begins with ρ and the second player with τ. The first player can create a random permutation ρ' for itself, and 120 permutations of the form $\tau' = \rho \circ \tau \circ \rho'^{-1}$ for each possible τ held by the second player. Oblivious Transfer (modified to be "1-120-OT") can give the second player the appropriate τ'.

In that same paper, Goldreich and Vainish present alternative implementations of both of these two-party functions which are secure against an adversary whose strength is slightly greater than that of a passive adversary. A "value-preserving" adversary may deviate from the protocol in any manner which does not affect the output of the computation. Protocols for both "and-xoring" and "permutation swapping" are given, and shown to be secure against a value-preserving adversary (under the QRA).

5.3 Multi-Party Cryptographic Protocols versus Active Adversaries

As described in Section 2.2.1, an active adversary can cause faulty players to deviate from a protocol arbitrarily. If a protocol is to be secure against such an adversary, then the honest players should be able to detect when deviations occur and to take corrective action. Often, the action to be taken is to kick out the offending players and continue the protocol somehow in their absence.

Channel:	broadcast only
Adversary:	t active, $t < n/2$
Security:	trapdoor function
Error Prob:	$O(2^{-k})$
bit complexity:	$O(B_Z(k)n^2C)$
round complexity:	$O(R_Z(k)n^2D)$

Table 3: Summary of Goldreich, Micali, and Wigderson 1987 (active adversary)

Goldreich, Micali, and Wigderson [57] show how to modify their multi-party private protocol to achieve a multi-party resilient protocol.

In the first stage of the protocol, each player commits to its secret input and to the random bits (generated by the community of players) that it will use during the computation phase. A verifiable secret sharing scheme is used to share the input with all players, and a coin-flipping scheme is used to generate random bits in a way that is reconstructible by the remaining players. After these commitments, any player can be fully "exposed" by the remaining community of honest players.

In the second stage, each message that is sent in the protocol is validated via zero-knowledge proof (as described in Section 2.1.5).

The protocol proceeds otherwise as in the passive adversary case. If any player is caught cheating, then its private input and random tape are reconstructed by the community (Recall that this is possible using the shares of the honest players). For the remainder of the protocol, any messages this player would have sent can be simulated privately by each remaining player.

Notice that in this model the penalty for cheating is severe. Banishment with full exposure is a reasonable response for a real cheater, but it may be a cruel and unusual reaction to an honest mistake. For example, if an honest player's messages are delayed in transit (stop failure), or otherwise due to some random distribution, then that player may find itself outside of the protocol with its secret input revealed to the community. In a kinder world, even a suspected cheater's secret input would remain secret, and rehabilitation (i.e., reentry into an ongoing protocol) would be possible.

Galil, Haber, and Yung [55] show how to modify the active adversary of Goldreich, Micali, and Wigderson to achieve this kinder world. The key idea is to use secondary shares (shares of shares) in the first stage of the protocol. After receiving shares of all secret inputs and random bits, each player verifiably shares all of these shares with the other players.

When a cheating player is discovered in stage two, then that player's secret input and random bits are not exposed. The primary shares held by that player are exposed, so that the remaining players can adjust their shares accordingly. However, whenever a message is needed from the cheating player, the other players compute it, in a secure distributed manner, from the shares they were given in stage one of the cheating player's input and random bits (thus "bootstrapping" secure computation to improve itself). Not only is the private input kept private, but a simple procedure allows the cheating player to rejoin the protocol at any time. In the rejoin protocol, each player splits each share in two. One is kept, and the other is given to the returning player to construct a share of its own.

Channel:	broadcast only
Adversary:	t active, $t < n/2$
Security:	discrete log for $n-1$, unconditional for 1
Error Prob:	$O(2^{-s})$
bit complexity:	$poly(n, s, C)$
round complexity:	$O(ns + D)$

Table 4: Summary of Chaum, Damgård, and Van de Graaf 1987

In Section 2.3.2, a two-party cryptographic protocol due to Chaum, Damgård, and Van de Graaf [36] was mentioned in which both players contributed to the "scrambling" of the circuit (truth table), and then both players evaluated the scrambled circuit at their private inputs. A more general version of this protocol works for any number of players. In the general case, each of n players contributes to the scrambling (i.e., each player takes a turn to further modify a "transform" of the circuit), and then all n players evaluate the scrambled circuit. This direct approach differs from the other solutions in this subsection, all of which reduce each gate to a collection of $O(n^2)$ two-party subprotocols. We present this protocol (modified for purposes of clarity) in some detail.

Recall from Section 2.3.2 that the method depends upon a bit commitment scheme with the additional properties of "blinding" (xor modification of commitments) and "comparability" (proof of equality of commitments).

To simplify the presentation, assume that the circuit consists of gates $g_1, g_2, \cdots, g_m$, each of which is either a binary AND gate or a unary NOT gate; in fact, the construction of Chaum *et al* evaluates arbitrary gates directly. As a further simplification, assume that each player i holds a single input bit x_i; the general construction can allow each player to hold any number of input bits. Lastly, assume that each input x_i is a 0-ary gate g_{x_i} with output x_i, $1 \leq i \leq n$, and that the output of the circuit is the output of the gate g_m.

1. A "0-transform" is constructed for each gate g in the circuit. The transform of a gate is a pair $[S, C]$ where C is a set of bit commitments, and where S is a set of tuples of bits and bit commitments. They are constructed as follows:

   (a) If g is a binary AND gate, then its 0-transform is $[S, C]$ where
   $S = \{< 0,0,0,\emptyset >, < 0,1,0,\emptyset >, < 1,0,0,\emptyset >, < 1,1,1,\emptyset >\}$ and $C = \emptyset$.
   (The first three components of each tuple in S give the truth table for AND.)

   (b) If g is a unary NOT gate, then its 0-transform is $[S, C]$ where
   $S = \{< 0,1,\emptyset >, < 1,0,\emptyset >\}$ and $C = \emptyset$.
   (The first two components of each tuple in S give the truth table for NOT.)

   (c) If g is a 0-ary input gate g_{x_i}, then its 0-transform is $[S, C]$ where $S = \emptyset$ and $C = \emptyset$.

2. For i from 1 to n, the ith player is assumed to have the $(i-1)$-transform of each gate in the circuit, and converts them into i-transforms as follows:

   (a) Choose $n + m$ random bits $c_{x_1}, \cdots, c_{x_n}, c_1, \cdots, c_m$, one for each gate in the circuit (including 0-ary input gates).

(b) For each gate g in the circuit, change the associated $(i-1)$-transform into an i-transform for the same gate as follows:

i. If $g = g_{x_i}$ is a 0-ary input gate, and the associated $(i-1)$-transform is $[S, C]$, then create a bit commitment for c_{x_i} and append it to C.

ii. If $g = g_{x_j}$ is a 0-ary input gate, $j \neq i$, then make no changes at all.

iii. If $g = g_l$ is a binary AND gate, then do the following:

A. Find the associated $(i-1)$-transform $[S, C]$ of g_l, where $S = \{< a_{11}, a_{12}, a_{13}, R_1 >, \cdots, < a_{41}, a_{42}, a_{43}, R_4 >\}$.

B. Find the gate g_{left} whose output wire connects to the left input wire of g_l, and replace each $(i-1)$-transform entry a_{j1} with $a_{j1} \oplus c_{left}$, $1 \leq j \leq 4$.

C. Find the gate g_{right} whose output wire connects to the right input wire of g_l, and replace each $(i-1)$-transform entry a_{j2} with $a_{j2} \oplus c_{right}$, $1 \leq j \leq 4$.

D. Replace each $(i-1)$-transform entry a_{j3} with $a_{j3} \oplus c_l$, $1 \leq j \leq 4$.

E. Create a bit commitment for c_l and append it to C.

F. Choose $4i$ random bits r_{jk}, $1 \leq j \leq 4$, $1 \leq k \leq i$. Let $r_j = r_{j1} \oplus r_{j2} \oplus \cdots \oplus r_{ji}$.

G. Replace each $(i-1)$-transform entry a_{j3} with $a_{j3} \oplus r_j$, $1 \leq j \leq 4$.

H. *Modify* the kth bit commitment in R_j so that it goes from being a commitment of b to being a commitment of $b \oplus r_{jk}$, $1 \leq j \leq 4$, $1 \leq k < i$. Notice that this step relies on the blinding property of the bit commitment.

I. Create a bit commitment for r_{ji} and append it to R_j, $1 \leq j \leq 4$.

J. Randomly permute the elements of S.

iv. If $g = g_l$ is a unary NOT gate, then do the following:

A. Find the associated $(i-1)$-transform $[S, C]$ of g_l, where $S = \{< a_{11}, a_{12}, R_1 >, < a_{21}, a_{22}, R_2 >\}$.

B. Find the gate g_{only} whose output wire connects to the only input wire of g_l, and replace each $(i-1)$-transform entry a_{j1} with $a_{j1} \oplus c_{only}$, $1 \leq j \leq 2$.

C. Replace each $(i-1)$-transform entry a_{j2} with $a_{j2} \oplus c_l$, $1 \leq j \leq 2$.

D. Create a bit commitment for c_l and append it to C.

E. Choose $2i$ random bits r_{jk}, $1 \leq j \leq 2$, $1 \leq k \leq i$. Let $r_j = r_{j1} \oplus r_{j2} \oplus \cdots \oplus r_{ji}$.

F. Replace each $(i-1)$-transform entry a_{j2} with $a_{j2} \oplus r_j$, $1 \leq j \leq 2$.

G. *Modify* the kth bit commitment in R_j so that it goes from being a commitment of b to being a commitment of $b \oplus r_{jk}$, $1 \leq j \leq 2$, $1 \leq k < i$. Notice that this step relies on the blinding property of the bit commitment.

H. Create a bit commitment for r_{ji} and append it to R_j, $1 \leq j \leq 2$.

I. Randomly permute the elements of S.

v. Player i uses a cut-and-choose zero-knowledge proof procedure to convince the other players that the i-transforms were created from the $(i-1)$-transforms correctly. This requires that the player create other legal i-transforms of the same truth table, and then respond to challenges by other players. In response to a challenge, one of the other i-transforms is either opened (i.e., all bit commitments revealed), or compared with the first i-transform (i.e., to show that either both are valid or both are invalid).

The comparison challenge is possible because of the "comparability" property of the bit commitment scheme. The details of this proof procedure are omitted.

At this point, the n-transforms of all gates have been constructed by player n, and are known to all players. The evaluation of the circuit at the players' input is as follows:

(c) Each player i announces its masked input bit $x_i \oplus c_{x_i}$.

(d) Mark every non-input gate $g_1, \cdots, g_m$ "unevaluated."

(e) While some gate remains unevaluated, do the following:

i. Choose an unevaluated gate g all of whose inputs have been announced.

ii. If g is a binary AND gate [NOT gate], then do the following:

A. Let $[S, C]$ be the n-transform for g, where $S = \{< a_{11}, a_{12}, a_{13}, R_1 >, \cdots, < a_{41}, a_{42}, a_{43}, R_4 >\}$ $[S = \{< a_{11}, a_{12}, R_1 >, < a_{21}, a_{22}, R_2 >\}]$.

B. Find the unique k, $1 \leq k \leq 4$ $[1 \leq k \leq 2]$, such that a_{k1} and a_{k2} match $[a_{k1}$ matches] the announced inputs [input] of g.

C. Open all bit commitments in R_k

D. Announce the output of the gate to be the exclusive-or of a_{k3} $[a_{k2}]$ and all committed bits in R_k.

iii. At this point, the output of gate g_m has been announced. Let $[S, C]$ be the associated n-transform of g_m.

iv. Open all bit commitments in C.

v. The output of the circuit is taken to be the exclusive-or of the announced output of g_m and all committed bits in C.

One interesting benefit of this protocol is that, if the bit commitment scheme unconditionally hides the committed bit (e.g., the discrete-log implementation described in Section 2.3.2), then one player's inputs are unconditionally secure, even against active cheating by the other $n - 1$ players. This property suggests that computation that hides *all* inputs unconditionally might be possible, a suggestion which Section 2.5 confirms. This property can also be used in a clever way to achieve a "hybrid" computation protocol [34], which protects both unconditionally (against a faulty minority) and cryptographically (against any faulty subset). The details of this hybrid are given in Section 2.5 as well.

5.4 Fairness in Cryptographic Multi-Party Protocols

The protocols described in this paper have thus far focused on the need to protect the secrecy of inputs while guaranteeing the correctness of the output. A different security property of interest is how to guarantee that interrupting a protocol confers no special advantage to any parties. This property is called "fairness."

The problem of "contract signing," allowing two parties to agree on a contract in a mutually committed fashion, predates and is related to the fairness problem; protocols for contract signing were given by Blum [23]. Fairness itself was originally considered for the basic problem of secret key exchange. If each of two players conveys a secret to the other, then one player can quit the protocol at any moment to advantage if at that moment it has received "more" than it has sent.

Blum introduced the problem [25], and gave an incorrect solution (generalized and corrected by Yao [94]) when the secrets were factorizations of large composite numbers. Each player takes turns revealing one bit of one prime factor, while proving that the bit is valid; being one bit "ahead" is of negligible advantage to a player who halts the protocol prematurely.

When the secrets are single bits, then no deterministic algorithms are known. A probabilistic solution was found by Luby, Micali, and Rackoff [76]: alternating flips of slightly biased coins gradually yield statistical information about the values of the secrets, and quitting leaves one player with the negligible advantage of knowing the outcome of at most one additional coin-flip. Vazirani and Vazirani [92] investigated a similar but somewhat weaker notion of exchanging a secret for a valid receipt.

Yao [95] first defined fairness for two-party computation protocols for boolean functions. Informally, a two-party protocol is fair if neither player can violate the protocol in such a way that the violator learns the output while the other player is unable to learn it. Galil, Haber, and Yung [55] extend the notion of fairness to multi-party computation, where it is called "synchrony." Suppose that there are n players and that there is an active adversary. The adversary has a certain advantage if it can halt the protocol at some moment when it has a computational advantage over the honest players at determining the output. If the adversary never can have such an advantage, then the protocol is said to be synchronous. Computational advantage can be defined to be an increased probability of successfully distinguishing the actual output from a possible output.

Galil, Haber, and Yung describe a means (similar to Blum [25], and following Yao [95]) for adding synchrony to a multi-party computation protocol, assuming that trapdoor functions exist. The n players jointly create a trapdoor key. Instead of performing a secure distributed computation of the actual function, the players securely compute the encryption of the output of the function using the trapdoor key. At this point, the players engage in a protocol to reveal the trapdoor key one bit at a time.

If the protocol is halted before the encryption of the output is found, then the adversary clearly has no advantage. If the protocol is halted during the gradual revelation of the key, then the adversary has only a negligible advantage of one bit over the honest players. This assumes, however, that the adversary and the honest players have equivalent computing power; otherwise, the adversary could halt when only it had enough bits of the trapdoor key to determine the result (e.g., by exhaustive search of all completions of the key).

Beaver and Goldwasser [13] present a different way of achieving fairness for boolean functions (similar to Luby *et al.* [76]), under the assumption that an Oblivious Transfer protocol exists (together with a network of private channels). It also does not rely on the computational equivalence of the adversary and the honest players. Instead of gradually revealing an encryption key that hides the output, what is gradually revealed is a series of values slightly biased toward the output.

The idea is to reveal a series of values, one at a time, which are related to the actual output. Each revealed value is the xor of the actual output and a random bit that is biased toward zero slightly (i.e., with inverse exponential advantage). The actual output can be inferred, with high probability, after seeing many such values. If the honest players quit after any detected violation, then the adversary can learn at most one more revealed value; this gives the adversary a negligible advantage for guessing the output. The protocols for creating the biased bits and for xoring the output can all be performed in a secure distributed manner. The biased bits are created by a coin-flipping protocol that relies on the existence of one-way functions; the existence of one-way

Channel:	complete private network, plus broadcast
Adversary:	t active, $t < n$
Security:	2-party oblivious transfer (e.g., noisy channels)
Error Prob:	$\frac{1}{p(n)}$, for any poly p
bit complexity:	poly(C, n)
roundcomplexity:	$O(D)$

Table 5: Summary of Beaver and Goldwasser 1989

functions is known to follow from the feasibility of oblivious transfer [15].

Goldwasser and Levin [59] extend the gradual revelation technique from boolean values to arbitrary values, again assuming only oblivious transfer. Representing the output as a boolean string, the computation stage reveals the bitwise xor of the output with a jointly created random mask. After this, the random mask is gradually revealed. This is done by gradually revealing, through slightly biased coin flips, a number of linear dependencies on the bits of the output. At the end of this process, each player learns enough linear dependencies to recover the output. Any premature termination leaves the adversary with only a negligible chance to guess one additional linear dependency.

Cleve [41] shows how to extend the secret exchange protocol of Luby *et al.* to allow for a "controlled disclosure" of information. Rather than having each party's guess of the other's bit merely converge to the correct result, the rate of convergence can also be specified. This can speed up the round complexity of those fair multi-party protocols that rely on this technique for gradual revelation.

6 Non-Cryptographic Secure Distributed Computation Protocols

6.1 Main Ideas

At first glance, the idea of a secure non-cryptographic protocol for the distributed computation of an arbitrary function seems paradoxical. How can the output of a function be computed by players of unbounded computational power without leaking *something*, even in only an information theoretical sense, about the private inputs? Indeed, the idea *is* paradoxical for arbitrary two-input functions computed by two mutually suspicious players, as was previously indicated (e.g., for two-party disjunction). Surprisingly, what is impossible in general for two players becomes possible when there are many players, by substituting secret-sharing, private channels, and "sufficient honesty" for cryptography.

An early result in this area (demonstrating the difference when more than two players participate) was due to Barany and Furedi [4], who give a 1-private non-cryptographic protocol for an aspect of the multi-player "Mental Poker" problem, i.e., how to deal a deck of cards among mutually untrustworthy players who can only communicate by sending messages. Their solution involves all players choosing random permutations of the deck, and sending messages consisting of

permutations applied to a player's currently held cards. If player p is to receive the next card, then these messages result in all other players learning the mapping of their current holdings under a permutation π known only to p. One of these other players can then select an unused element in the range of π, and its inverse will be the card next dealt to p.

Most subsequent protocols for non-cryptographic secure computation, and all of the ones given in this section, rely on the secret-sharing method due to Shamir [88]. This method was described in Section 2.1.6: To "t-share" a secret s, give each player i a point $(\alpha_i, p(\alpha_i))$ for some predetermined $\alpha_i \neq 0$ (e.g., $\alpha_i = i$), where $p(x)$ is an otherwise random degree t polynomial whose constant term is s. All non-cryptographic computation protocols described in this paper use this method in a "three-stage paradigm" [57]. First, each player shares his secret input with all other players using a Shamir polynomial. Second, all players perform some computation on these shares. Third, the results of this computation are combined to find the actual output.

Nice homomorphic properties of Shamir shares (first pointed out by Benaloh [17]) make stage two possible: any linear combination of shares of secrets is itself a share of the linear combination of secrets. For example, if $p(x)$ and $p'(x)$ are otherwise random degree t polynomials with constant terms s and s', then $p(x) + p'(x)$ is an otherwise random degree t polynomial with constant term $s + s'$. Moreover, if player i holds shares $s_i = p(\alpha_i)$ and $s'_i = p'(\alpha_i)$, then $s_i + s'_i$ is his share of the sum. The same can be shown for the product of a secret by a scalar.

A "close call" prevents stage two from being entirely non-interactive, i.e., the product of secret shares is "almost" a share of the product of the secrets. If $p(x)$ and $p'(x)$ are otherwise random degree t polynomials with constant terms s and s', then $p(x)p'(x)$ does have constant term ss', but it is neither degree t (i.e., it is degree $2t$) nor otherwise random (e.g., it cannot be irreducible). Both of these flaws are critical. The larger the degree of the share polynomial, the more players that are needed to recover the secret, so it is essential that the degree not increase with each multiplication. If the share polynomials are not fully random, then information about the secrets may be leaked to fewer than $t + 1$ computationally unbounded agents.

One of the main trends of the non-cryptographic computation protocols described in this paper follow from this "close call." Computation in Stage 2 proceeds directly on shares of secrets, with occasional interaction to "clean up" shares after multiplication.

Another main trend arises from this first one when the adversary is active. If interaction is required for multiplications, then how can the honest players protect themselves against misleading interaction coordinated by an active adversary? The solutions to this problem depend on the level of resilience that is required. If fewer than one-third of the players are faulty, then error-correcting codes can be exploited [20]. For greater resilience, new techniques of verifiable secret sharing [85] can be used. These new techniques can require more interaction among players, even for linear operations.

In this subsection, we will survey some of the non-cryptographic secure distributed computation protocols that have been developed. As in the previous section, some protocols are accompanied by a table that summarizes important features. In these tables, the number of players in the protocol is denoted by n, the size of a circuit is denoted by C, and the depth of a circuit is denoted by D. All protocols in this section provide computation over a finite field (usually $GF(p)$ for some prime p, occasionally $GF(2^k)$, and rarely $GF(p^k)$ for some prime $p \neq 2$); we assume that it takes $\log |F|$ bits to communicate an element of the finite field F.

Channel:	complete private network
Adversary:	t passive, $t < n/2$
Security:	unconditional
bit complexity:	$O(n^2 C \log \lvert F \rvert)$
round complexity:	$O(D)$

Table 6: Summary of Ben-Or, Goldwasser, and Wigderson 1988 (passive adversary)

6.2 Non-cryptographic Protocols versus Passive Adversaries

The first two papers suggesting general non-cryptographic distributed computation are by Ben-Or, Goldwasser and Wigderson [20], and by Chaum, Crépeau, and Damgård [35]. Both papers present protocols for t-private computation whenever $n > 2t$. Provided that all players obey the protocols perfectly, no minority of players can pool their knowledge at the end of the protocol to gain further information about the honest players' inputs. In this subsection, we will describe both of these protocols.

Ben-Or, Goldwasser, and Wigderson [20] present a protocol which allows t-private computation among $2t + 1$ players without any cryptographic assumptions. A network of private channels connecting all players is required.

This protocol, as with all protocols developed thus far for the non-cryptographic setting, is based on Shamir's method for secret sharing. Each player i shares its secret input s_i with the other players by giving each other player j a single point $p_i(\alpha_j)$ on an otherwise random degree t polynomial $p_i(x)$ such that $p_i(0) = s_i$. For simplicity, assume that $\alpha_i = i$ for all i, $1 \leq i \leq n$. Any arbitrary linear combination $\lambda(s_1, \cdots, s_n)$ of secret inputs can then be performed by the players without any communication, as described at the start of this section.

Since arbitrary linear computations can be performed directly on t-shares without communication, it suffices to show how multiplication of shared secrets can be performed. For this operation, some communication is needed. If each player multiplies together its shares of the multiplicands, then, analogous to the linear case, each player arrives at a value on a polynomial that passes through the product at the origin. However, this polynomial that "almost-shares" the product is neither random nor of the right degree. It is a degree $2t$ polynomial that has a certain structure, e.g., it cannot be irreducible. The clever subprotocol for multiplication uses communication to eliminate this structure.

Suppose that the multiplication subprotocol begins with each player i holding the share $p(i)$ of the secret $s = p(0)$ and the share $p'(i)$ of the secret $s' = p'(0)$, where p and p' are otherwise random degree t polynomials. Let B be the n by n (vandermonde) matrix whose (i, j) entry is j^i. Let $Chop_t$ be the n by n matrix whose (i, j) entry is 1 if $1 \leq i = j \leq t$ and 0 otherwise. The subprotocol is as follows:

1. Each player i privately finds an otherwise random degree t polynomial $r_i(x)$ with constant term zero.

2. Each player i sends to each player j the value $r_i(j)$.

Channel:	complete private network
Adversary:	t passive, $t < n/2$
Security:	unconditional
bit complexity:	$O(n^2 C \log \lvert F \rvert)$
roundcomplexity:	$O(D)$

Table 7: Summary of Chaum, Crépeau, and Damgård 1988 (passive adversary)

3. Each player i privately finds $v_i = p(i)p'(i) + \sum_{j=1}^{n} r_j(i)$.

4. The players cooperate to give each player i the ith component of $(v_1 v_2 \cdots v_n) B^{-1} Chop_t B$, as follows:

   (a) Each player i privately finds an otherwise random degree t polynomial $q_i(x)$ such that $q_i(0) = v_i$.

   (b) Each player i sends to each player j the value $v_{ij} = q_i(j)$.

   (c) Each player i privately finds $(w_{1i} w_{2i} \cdots w_{ni}) = (v_{1i} v_{2i} \cdots v_{ni}) B^{-1} Chop_t B$

   (d) Each player i sends to each player j the value w_{ji}.

   (e) Each player i privately interpolates to find the value at 0 of the degree t curve through $(1, w_{i1}), (2, w_{i2}), \cdots, (n, w_{in})$. This is the desired share at i of the product of s and s'.

Notice that this subprotocol for multiplication first re-randomizes the intermediate degree $2t$ polynomial (by adding random polynomials with zero constant term), and then truncates its degree down to t (by multiplying by $B^{-1} Chop_t B$. Since this truncation step is itself a linear operation, it can be computed privately on shares of the "almost-shares" (step 4c).

Chaum, Crépeau, and Damgård present a protocol that also allows t-private computation among $2t+1$ players without any cryptographic assumptions, also assuming a network of private channels connecting all players. In fact, this protocol has some similarities to that of Ben-Or, Goldwasser, and Wigderson, although the computation proceeds bit by bit here, on a circuit composed of AND gates and XOR gates over $GF(2^k)$.

Assume that each player has some input bits, and that the function to be computed is represented as a circuit composed of AND gates and XOR gates. Each player t-shares his input bits using Shamir polynomials over a finite field that is a power of two (i.e., $GF(2^k)$ for some k such that $2^k > n$). Then XOR in this protocol becomes analogous to addition in the previous protocol, while AND becomes analogous to multiplication.

Each XOR gate can be computed privately without any interaction. Each player simply adds its t-shares of the corresponding inputs, and arrives at a valid t-share of the output. Notice that the new share lies on a polynomial that passes through the XOR at zero only because the underlying field was chosen to be a power of two.

Each AND gate can "almost" be computed by having each player multiply its two t-shares of the corresponding inputs. As before, the resulting values are on a polynomial that passes through zero at the right place, but which is non-random and of twice the desired degree. Chaum, Crépeau,

Channel:	complete private network
Adversary:	t active, $t < n/3$
Security:	unconditional
Error Prob:	zero
bit complexity:	$O(n^5 C \log \lvert F \rvert)$
roundcomplexity:	$O(D)$

Table 8: Summary of Ben-Or, Goldwasser, and Wigderson 1988 (active adversary)

and Damgård have an interesting method for the interactive "re-randomization" and "truncation" steps for repairing these values, doing both of these steps simultaneously.

After the private multiplication, when each player has an "almost-share" of the AND result, the players proceed as follows. Each player chooses two otherwise random polynomials, one of degree t ("low-degree") and one of degree $2t$ ("high-degree"), such that either both have constant term zero or both have constant term one. Each player distributes shares of both polynomials. Then each player adds its "almost-share" of the AND computation to the sum of the high-degree shares it has just received. This yields valid $2t$-shares of an XOR of random values with the AND result. The players reveal these shares and determine what this XOR is. If it is zero, then the XOR of the random values is equal to the AND result; in this case, the sum of the low-degree shares for each player is the valid t-share of the AND result. If it is one, then the XOR of the random values is the opposite of the AND result; in this case, one plus the sum of the low-degree shares for each player is the valid t-share of the AND result.

6.3 Non-cryptographic Protocols versus Active Adversaries

In this subsubsection, we will cover four non-cryptographic protocols for distributed computation versus an active adversary. The first two [20] [35] achieve t-resilience whenever there are $n > 3t$ players, assuming a complete private network of channels. The last two [86] [9] achieve t-resilience when $n > 2t$, but require a broadcast channel in addition to a complete private network, and must allow a small probability of error.

In the same paper that gave the protocol for t-private computation when $n > 2t$ (described in the preceding subsection), Ben-Or, Goldwasser and Wigderson [20] also demonstrated that t-resilient computation was possible for $n > 3t$. The resilient protocol is identical to the private protocol except for changes to the secret-sharing method and to the multiplication stage.

The verifiable secret sharing scheme outlined in Section 2.1.6 is used in place of simple secret sharing. Furthermore, instead of having player i receives values at some arbitrary α_i (e.g., $\alpha_i = i$), this protocol requires that each player receive values at a specific nth root of unity. For example, once a player begins the VSS protocol by choosing a polynomial $f(x, y)$ whose constant term is the secret, then each player i receives the polynomials $f(x, \omega^i)$ and $f(\omega^i, y)$, where ω is a predetermined primitive nth root of unity (and where the actual share for player i is $f(\omega^i, 0)$).

The importance of this choice for share locations is that the vector of n shares of any secret is a *codeword* of a generalized Reed-Muller error-correcting code [22]. This family of codes can correct a codeword in which up to one-third of the components are incorrect or missing. For our

purposes, this means that the $n - t$ honest players can reconstruct a shared secret after t arbitrary shares are corrupted by an active adversary. Furthermore, the correction procedure involves only the computation of $2t$ *linear* combinations of codeword components (called "syndromes"); thus correction may be performed privately on shares of codewords.

The multiplication phase differs from the private protocol in two ways. Re-randomization requires that each player distribute shares of an otherwise random degree $2t$ polynomial with constant term zero. For purposes of resilience, the honest players must be able to prove that the shares that are received are in fact from polynomials of this form. There is a protocol for doing this, based on the fact that degree t polynomials can be so verified, and that a share of an otherwise random degree $2t$ polynomial with zero constant term can be constructed privately from shares of t random degree t polynomials. Beaver [9] has observed, however, that this extra effort to randomize the higher-degree coefficients is wasted, since these will disappear after truncation anyway.

The second difference is in the truncation step. A subprotocol is inserted to force all players to "validate" their inputs to the degree truncation step. Recall that each player shares its share of the product, so that a linear combination of the first-level shares can be performed privately by each player on the second-level shares. The honest players need to be sure that these second-level shares are in fact honest shares of the first-level shares.

Suppose that each player i begins a multiplication phase with the t-shares $p(\omega^i)$ and $p'(\omega^i)$ of the original multiplicands $p(0)$ and $p'(0)$. After re-randomization, suppose that each player i has a $2t$-share $q(\omega^i)$ of the product $q(0) = p(0)p'(0)$.

Each player i begins by distributing t-shares of $p(\omega^i)$ and of $p'(\omega^i)$. This gives each player a share of two codewords. Using the linear error correction procedure for this coding scheme, each player can compute shares of the $2t$ syndromes for the two codewords. The players reveal these shares, reconstruct the syndromes, and use the syndromes to find the right value for each incorrect $p(\omega^i)$ or $p'(\omega^i)$. For each such incorrect value, each player can replace the corresponding share with the correct value itself (essentially forcing that player to have used the constant polynomial to share its share).

At this point, all honest players are convinced that all shares of shares of $p(0)$ and $p'(0)$ are correct. Now each player needs to distribute t-shares of its $2t$-share of the actual product $p(\omega^i)p'(\omega^i)$, in such a way that the honest players are convinced that the proper product relation in fact holds among the three shares. This can be done by private computation on shares of $t + 1$ (verifiably) degree t polynomials, in a manner similar to that used in the re-randomization step.

Chaum, Crépeau and Damgård [35] also give a protocol for t-resilient computation whenever $n > 3t$. There are several important similarities with the protocol of Ben-Or *et al.* First, this protocol also adapts a privacy-only protocol by overlaying verifiability subprotocols. Second, an important part of both protocols is that players operate on shares of shares of secrets. This is a technique that was first used in the cryptographic setting by Galil, Haber, and Yung [55] to tolerate faults without revealing inputs of faulty parties.

There are also some significant differences between the two solutions. The protocol of Chaum, Crépeau, and Damgård relies on an interactive "cut-and-choose" procedure to validate that all of the shares of a secret are consistent with one another, i.e., that all n values lie on a single degree t polynomial. Unlike the protocol of Ben-Or *et al.*, which used error-correcting codes for this purpose, this leaves an exponentially small probability of undetected cheating. This also makes the protocol less efficient in terms of number of rounds and messages.

Channel:	complete private network, plus broadcast
Adversary:	t active, $t < n/2$
Security:	unconditional
Error Prob:	$O(2^{-k})$
bit complexity:	$O(n^6 k^3 \|F\|^4 \log \|F\|)$
round complexity:	$O(D)$

Table 9: Summary of Beaver 1989 / Rabin and Ben-Or 1989

Since Byzantine Agreement requires that more than two-thirds of the players be honest, these two resilient protocols [20] [35] are the best possible for a network of private channels in the non-cryptographic setting. One natural question is whether the resilience can be increased by adding a broadcast channel.

The affirmative answer was supplied T. Rabin and Ben-Or [86], and by Beaver [9], both building on ideas from T. Rabin [85].

At the heart of the protocol is a verifiable secret sharing scheme, due to T. Rabin [85], that can correct up to t errors where $n > 2t$, by giving up on absolute certainty of success. This scheme has a probability of failure that is exponentially small with respect to a security parameter. This VSS scheme also requires the existence of a broadcast channel.

One key idea in the VSS method is that of "information check vectors." This is a primitive for message sending involving three parties: a dealer D, a receiver R, and an intermediary INT. The basic protocol is as follows (with computation over $GF(p)$ for some prime p). If D has a secret s to send to R, then D begins by choosing random b and y. D then sends the pair (s, y) to INT, and D sends the pair $(b, c = s + by)$ to R. At a later moment, INT forwards to R the pair (s, y) received from D. R accepts the message s if and only if $c = s + by$.

Using check vectors, a weaker form of secret sharing (called "weak secret sharing") can be implemented. This is a protocol that behaves like verifiable secret sharing when the dealer is honest, and which provides limited protection against a dishonest dealer. The secondary weak sharing of Shamir shares is part of Rabin's VSS scheme.

Because of the secondary sharing, Rabin's VSS scheme does not have the same homomorphic properties as Shamir's basic secret-sharing scheme, i.e., shares of linear combinations of secrets cannot be determined through private computation on the original shares. However, with interaction, Rabin shows that there is a t-resilient protocol to give each of $n > 2t$ players the appropriate share of any linear combinations of secrets.

Using the protocol for linear combination, a protocol is given to t-resiliently demonstrate that one secret c is the product of two other secrets a and b. In other words, if a, b, and c have each been shared by Rabin's VSS scheme by one player, and if $c = ab$, then that player can convince the honest players of this fact without revealing anything further about the three secrets. This is done by a type of cut-and-choose protocol. A number of equalities related to the original secrets, of the form $D = (a + R)(b + S)$, are chosen by the verifier; for each such equality, the values R, S, and D are verifiably shared; some of these triples are challenged; those that are challenged are revealed to demonstrate that the corresponding equalities are indeed correct; those triples that are

Channel:	complete private network
Adversary:	t active, $t < n/3$
Security:	unconditonal
Error Prob:	zero
bit complexity:	poly$(C, n, \|F\|)$
roundcomplexity:	$O(1)$

Table 10: Summary of Bar-Ilan and Beaver 1989

not challenged are combined with the original three secrets to demonstrate that the equalities are indeed related to the original secrets. The probability of undetected cheating is bounded by 2^{-k}, where k is the number of triples that are used by the verifier.

The t-resilient protocol for linear combination also implies a t-resilient protocol for degree truncation. In other words, if each player holds a share of a degree $2t$ polynomial $p(x)$, then this protocol ends with each player holding a corresponding share of a degree t polynomial $\overline{p}(x)$ with the same coefficients for all of its terms. Each output share is in fact a specific linear combination of the input shares: $\overline{p}(\alpha_i) = \sum_{i=i}^{n} \overline{L_i}(\alpha_i)p(\alpha_i)$, where $\overline{L_i}(x)$ is the (publicly computable) degree t truncation of the Lagrange polynomial which is one at α_i and zero at α_j, $j \neq i$.

Re-randomization is straightforward, due to Beaver's observation [9] that the only coefficients that need to be randomized are those that will survive the truncation step. Each player i can verifiably t-share a random value using a polynomial $p_i(x)$. If each player j now multiplies each received share $p_i(\alpha_j)$ by its own share location α_i, then it obtains a valid $(t+1)$-share of zero. The sum of all of these $(t+1)$-shares is a point on an otherwise random degree $t+1$ polynomial with constant term zero.

Using the t-resilient protocols for product demonstration, re-randomization, and degree truncation, a t-resilient protocol for multiplying two shared secrets is possible. First, each player verifiably t-shares three values: its shares of the two secrets, and the product of its shares of the two secrets. Second, each player t-resiliently demonstrates that the proper product relation holds among these three shared values. Finally, the players perform the t-resilient protocols for re-randomization and truncation.

6.4 Constant-Round Non-cryptographic Computation

One of the open problems in secure distributed computation is whether $O(n)$-resilience is possible in constant rounds and polynomial-sized messages in the non-cryptographic setting. This section presents some of what is currently known about constant-round non-cryptographic computation versus an active adversary.

Bar-Ilan and Beaver [5] demonstrate that, even in the absence of cryptographic assumptions, the number of rounds of communication needed for secure distributed computation of a function need not be related to the depth of the arithmetic circuit for that function. Bar-Ilan and Beaver show how to compute any NC^1 circuit in constant rounds and polynomial-sized messages.

One of the key ideas in this protocol is the use of a technique, due to Ben-Or and Cleve [19], for mapping an arbitrary arithmetic function to the multiplication of a string of three by three

Channel:	complete private network
Adversary:	t active, $t < c \log n$
Security:	unconditional
Error Prob:	zero
bit complexity:	poly$(n, \log \lvert F \rvert)$
roundcomplexity:	$O(1)$

Table 11: Summary of Beaver, Feigenbaum, Kilian and Rogaway 1989

matrices. By a self-randomization trick (similar to the method used by Kilian for the two-party case [70]), this long product of matrices can be securely computed using only two secret matrix multiplications:

$$Y = M_1 M_2 \cdots M_N = R_0({R_0}^{-1} M_1 R_1)({R_1}^{-1} M_2 R_2) \cdots ({R_{N-1}}^{-1} M_N R_N) {R_N}^{-1}$$

$$= R_0 Z {R_N}^{-1}.$$

Thus, if the players jointly create and share random invertible matrices $R_0, \cdots, R_N$, they can compute each $S_i = {R_{i-1}}^{-1} M_i R_i$ in parallel by private computation, public revelation, and interpolation. From these they can find $Z = S_1 S_2 \cdots S_N$, and then two secret multiplication will find $Y = R_0 Z {R_N}^{-1}$.

When f is not NC^1, the function can still be securely computed in constant rounds, but with messages that are not necessarily polynomial in size. Alternatively, using their technique, an arbitrary function can be securely computed in $D/\log n$ rounds, where D is the circuit depth, using polynomial-sized messages.

Beaver, Feigenbaum, Kilian, and Rogaway [12] show how to achieve non-cryptographic computation of boolean circuits of any size in constant rounds with polynomial-sized messages, by decreasing the resilience to $O(\log n)$ faulty processors.

The method relies on an instance-hiding scheme for any boolean function using $n/\log n + 1$ oracles. This can be converted into an $n+1$ oracle instance-hiding scheme that is $(\log n)$-private, by replacing each oracle query at x by $n+1$ oracle queries at $(\log n)$-shares of x.

The method also relies on the constant-round poly-sized computation of functions with log-depth circuits [5]. Creation of appropriate oracle queries, polynomial interpolation, notarized envelope schemes, and majority-voting all have log-depth circuits, and so all are available as subprotocols in this way.

Lastly, Beaver, Micali, and Rogaway [14] show that constant-round poly-sized computation, resilient for a constant fraction of faulty processors, is achievable assuming that one-way functions exist. The protocol, a generalization of the two-party protocol due to Yao [95] described in Section 2.3.2, has the players construct a scrambled version of the circuit, from which each player can compute the output on its own. Although the scrambled circuit is consistent with only one set of values of the secret inputs, conditional privacy is established through the use of a pseudorandom generator (i.e., one-way function) to hide connections between scrambled gates. Because of the reliance on one-way functions (along with a broadcast channel and a network of private channels),

Channel:	private network, plus broadcast
Adversary:	t active, $t < n/3$
Security:	one-way function (i.e., pseudorandom generator)
Error Prob:	$O(2^{-k})$
bit complexity:	$\text{poly}(C, n, k, \log\|F\|)$
roundcomplexity:	$O(1)$

Table 12: Summary of Beaver, Micali, and Rogaway 1990

Channel:	complete private network
Adversary:	c passive, d active, $n > 2d + c$, $n > 2c$ (or cryptosystem secure)
Security:	uncond if $t < n/2$; trapdoor perm + claw-free func if $n/2 \leq t < n$
Error Prob:	$O(2^{-k})$
bit complexity:	$\text{poly}(C, n, k, \log\|F\|)$
roundcomplexity:	$O(D)$

Table 13: Summary of Chaum 1989

this is actually a *cryptographic* protocol. However, its relevance is clearer when mentioned along with other constant-round results.

In addition to one-way functions, this protocol requires a broadcast channel and a network of private channels, and allows an exponentially small probability of error. The only use of one-way functions is to guarantee the existence of pseudorandom generators, an equivalence shown by Impagliazzo, Levin and Luby [66] and Håstad [63].

6.5 Hybrid Computation Protocols

Chaum [34] combines both non-cryptographic and cryptographic computation into a single hybrid protocol. The security of this protocol is compromised only if *both* a majority of the players are faulty *and* the underlying cryptographic assumptions are violated.

The key idea is to transform the n-ary function evaluation $f(s_1, \cdots, s_n)$ into an $(n+1)$-ary function evaluation $f^*(s_1 \oplus r_1, \cdots, s_n \oplus r_n, r_1 \circ \cdots \circ r_n))$, where $\circ$ denotes concatenation. Here each player i chooses the random r_i. The computation of f^* is performed using the cryptographic protocol of Chaum, Damgård, and Van de Graaf [36] (see Section 2.4.3), where all n players jointly determine all messages sent by player $n+1$. They do this by running a non-cryptographic protocol (e.g., [35]) whenever a message from this "player" is needed.

Recall from Section 2.4.3 that the cryptographic protocol of Chaum, Damgård, and Van de Graaf can protect one player's input unconditionally; here this protection is granted to the jointly maintained player. Thus breaking the outer cryptographic protocol reveals only the first n inputs, each of which is a secret xored to a random value. To determine these random values requires breaking the inner non-cryptographic protocols as well. Alternatively, breaking only the inner

non-cryptographic protocol reveals only information about the random values, which yields no information about the original secrets.

6.6 Non-Cryptographic Computation Protocols on Incomplete Networks

All of the non-cryptographic protocols described in this subsection thus far have required that the network of private channels connecting the players be complete. It is interesting to consider what weaker assumptions on the private network will still allow secure distributed computation without cryptographic assumptions.

Rabin and Ben-Or [86] show how to perform any secure distributed computation t-resiliently in a network of at least $3t + 1$ players, if the players are joined by a network of private channels with connectivity at least $2t + 1$. This is done by showing how to send a message from any player to any other player through such a network, in expected time equal to the diameter of the network, and with a slight probability of error. This capability can be spliced into previously discussed protocols to achieve the stated result.

Dolev, Dwork, Waarts, and Yung [48] show how to attain the same result without any probability of error. Sending a message from any player to any other player ("perfectly secure message transmission") is divided into two problems: the slightly easier problem of sending a one-time random pad, and the much easier problem of sending the message xored with the one-time pad. The second problem just requires that the encrypted message be sent simultaneously along all $2t + 1$ paths, whereupon the receiver decrypts the majority message. The first problem requires that each bit of random pad be transmitted via an interactive protocol. To transmit one random bit, a three-phase protocol is used in which shares of $nt + 1$ bits and associated checking pieces are sent along different paths from sender to receiver. No amount of cheating by the t faulty players can prevent the sender and receiver from agreeing on one of the $2t + 1$ random bits that is guaranteed to be uncorrupted, or allow the faulty players to learn the value of that bit.

6.7 Non-Cryptographic Protocols versus Mobile Adversaries (Virus Model)

Ostrovsky and Yung [83] consider a more powerful adversary that models the behavior of a (detectable) virus in a computer network (with rebootable machines): a different set of up to t players can be under the adversary's control each round. They present a non-cryptographic computation protocol secure against such an adversary where t is some constant fraction of the number of players. One of the features of this protocol is that all of the information held secret by the players must be continually reshared, so that the mobile adversary never gets enough consistent shares of any secret to recover it.

6.8 Non-Cryptographic Asynchronous Computation

Ben-Or, Canetti and Goldreich [16] study secure computation among processors connected by a complete *asynchronous* network of untappable channels. If only crash failures are allowed (Fail-Stop faults), they show how to compute securely any function over a finite field while withstanding up to $\lceil \frac{n}{3} \rceil - 1$ faulty processors. If arbitrary failures are allowed, they show how to compute any function over a finite field $(\lceil \frac{n}{4} \rceil - 1)$-resiliently. One of the tools they use for these results is a verifiable secret sharing scheme in the asynchronous model for $n \geq 4t + 1$, with zero probability of error.

7 Reductions and Complexity Results

Previous subsections have contained descriptions and discussion of what is possible for secure distributed computation. This subsection will survey what is known about reductions and complexity results for this problem, i.e., what is just as hard as, and what is impossible for, secure distributed computation.

7.1 Reductions Among Primitives

Kilian [71] shows how to base two-party oblivious circuit evaluation on a black box for Oblivious Transfer. A sketch of this important reduction was given in Section 2.3.2. In later work [72], Kilian extends the reduction by establishing a necessary and sufficient condition for basing two-party oblivious circuit evaluation on a black box for securely computing some 2-ary function F. Specifically, he proves that Oblivious Transfer can be implemented from a black box for securely computing F if and only if F possess an "imbedded OR," i.e., that there exists $i_0, i_1, j_0, j_1, x_0, x_1$ such that for all $a, b \in \{0,1\}$, $F(i_a, i_b) = x_{a \vee b}$.

Ostrovsky, Venkatesan, and Yung [81] consider the problem of sufficient conditions for various "asymmetric" two-party protocols, i.e., protocols in which one player is polynomially bounded while the other player is computationally unbounded. In particular, they show that Oblivious Transfer is possible, in either direction, assuming the existence of one-way functions. Since Kilian [70] reduces two-party circuit evaluation to Oblivious Transfer, it follows that any asymmetric two-party computation is possible if one-way functions exist.

The idea of the results of Ostrovsky, Venkatesan, and Yung may be conveyed by considering a weaker version of one direction: Oblivious Transfer from an unbounded player to a bounded player under the assumption that one-way *permutations* exist. The weaker player chooses a random n-bit element x' and finds $x = \sigma(x')$, where σ is a one-way permutation. The weaker player conveys to the stronger player $n-1$ bits of information, one at a time, about the range value (by answering $n-1$ queries for inner products of random strings with x); each answer reduces the domain of initial choices, ultimately arriving at a set containing the real choice and one additional possible value. The stronger player now guesses at random among the two possible remaining choices for x, and inverts the one-way permutation at that guessed value. This inverted value is either x' or some value the weaker player cannot compute. By hiding his secret using the inverted value (i.e., by xoring it with an inner product of the inverted value and a public random value), the weaker player can recover the secret exactly half the time.

For the symmetric case, Impagliazzo and Luby [67] show that one-way functions are necessary for bit commitment, and hence necessary for secure computation. This paper also shows that Oblivious Transfer implies the existence of one-way functions. However, due to a relativization result from Impagliazzo and Rudich [68], it is unlikely that the sufficiency of one-way functions for secure symmetric computation can be shown (i.e., a proof that only used one-way functions in a "black-box" manner would prove $P \neq NP$).

Ostrovsky and Yung [82] demonstrate the necessity of a complete network of private channels for implementing private multiparty computation among computationally unbounded players (sufficiency was shown by [20] and [35]). They consider the problem of dealing one card from a four card deck to each of three players. If one of the channels between two players is insecure (i.e., accessible to the third player), then some information about the distribution of the cards must be

leaked.

7.2 Necessary Conditions and Resources

The first complexity results in this field are two impossibility results due to Yao [94], one for a certain type of two-player secret exchange (swapping zeros of trapdoor functions), and the other for the generation of an arbitrarily biased bit by many players.

Ben-Or, Goldwasser, and Wigderson [20] provide matching lower bounds for multiparty non-cryptographic computation on a network of private channels, for both passive and active adversary. For a passive adversary, there are functions for which there is no t-private protocol among n players when $n \leq 2t$; e.g., the impossibility of a 1-private protocol for two parties to compute the OR of two bits is easy to verify. For an active adversary, there are functions for which there is no t-private protocol among n players when $n \leq 3t$; this follows directly from the lower bound for Byzantine Agreement [74].

For the case of a passive adversary, more is known about when a function does or does not have a t-private protocol among n players when $n \leq 2t$. Chor and Kushilevitz [47] find a "gap" in the maximum level of privacy, in the non-cryptographic zero-error setting for boolean functions. If a function has a t-private protocol, $n \leq 2t$, then it has an $(n-1)$-private protocol. Moreover, every such n-ary function is equivalent to an xor of n single-input functions.

For the case of an active adversary, Cleve [40] has impossibility results for the much simpler problem of computing a single random bit. When at least half of the processors are faulty, it is impossible to compute a random bit in polynomial time with a negligible bias (less than the reciprocal of any polynomial in the number of players).

For two player non-cryptographic protocols, a complete characterization of privately computable general functions (i.e., non-boolean) is given independently by Kushilevitz [73] and by Beaver [8]. A function is privately computable if and only if it is "partitionable." Consider the tabular form of a two-input function $f(x, y)$, with each row corresponding to a different possible value for x, each column to a different possible value for y, and each entry representing the value of f at those inputs. Call a function "column-partitionable" if the columns of the table can be divided into two subsets such that no range value appears in both parts; define "row-partitionability" similarly. Consider that either kind of split turns one function into two functions, each defined over part of the domain of the original. A function is "partitionable" if it is constant (all table entry are the same), or if it is row-partitionable into two partitionable functions, or if it is column-partitionable into two partitionable functions.

For example, the function $f(x, y) = \max(1, x^2 \bmod 5, y)$ defined on the domain $\{0, 1, 2, 3\} \times \{0, 1, 2, 3\}$ is partitionable, and hence privately computable (see Figure 1). However, the function $g(x, y) = \min(1, x^2 \bmod 5, y)$ defined on the same domain is not partitionable, and hence not privately computable (see Figure 2).

Bar-Yehuda, Chor, and Kushilevitz [7] consider two-party "nearly private" computation protocols for functions that are not partitionable; a "nearly private" protocol leaks some information about private inputs to the opposing player. Their formal definition of information leakage is similar to the notion of information leakage used by Abadi, Feigenbaum, and Kilian [1] for characterizing instance-hiding schemes. Bar-Yehuda, Chor, and Kushilevitz show that the identity function and the greater-than function can be computed leaking at most $\log n$ bits of information, and that almost all boolean functions can be computed leaking at most $\frac{1}{2}(n - \log n - 3)$ bits of informa-

x/y	0	1	2	3
0	1	1	2	3
1	1	1	2	3
2	4	4	4	4
3	4	4	4	4

Figure 1: Tabular form of $f(x, y) = \max(1, x^2 \bmod 5, y)$ on $\{0, 1, 2, 3\}^2$

x/y	0	1	2	3
0	0	0	0	0
1	0	1	1	1
2	0	1	1	1
3	0	1	1	1

Figure 2: Tabular form of $g(x, y) = \min(1, x^2 \bmod 5, y)$ on $\{0, 1, 2, 3\}^2$

tion. They also construct functions for which large gains in round complexity are achievable by leaking small amounts of information; for one artificial example, exponential round complexity can be reduced to two rounds and linear number of bits of communication by revealing a single bit of information about the inputs.

All of the results in this paper assume that the computation is performed over some finite field. Chor, Gereb-Graus, and Kushilevitz consider the case of private multi-party non-cryptographic computation over countable domains. They show that every n-ary boolean function is either n-private, $\lfloor \frac{n-1}{2} \rfloor$-private but not $\lceil \frac{n}{2} \rceil$-private, or not 1-private. Their results follow from establishing a connection between private computability and communication complexity as defined by Yao [93]. As a surprising special case, private addition over a finite subrange of integers can be n-private, over the positive integers can be at most $\lfloor \frac{n-1}{2} \rfloor$-private, but over all the integers cannot even be 1-private.

Dolev, Dwork, Waarts, and Yung [48] give lower bounds on security versus connectivity for the simpler but related problem of perfectly secure message transmission through an untrusted network. They consider this problem in a setting that generalizes in two ways on the setting considered for the multi-party unconditional protocols in this paper: (1) there are two adversaries, one passive and one active, which control possibly overlapping sets of faulty processors, and which may or may not be able to communicate during execution; and (2) they assume an incomplete network, of known degree.

8 Discussion and Open Problems

In the past few years, much progress has been made in understanding the problem of secure distributed computation. General-purpose protocols have been constructed for a wide variety of settings, including many that achieve the provably best bound for some resource. Progress toward a

satisfactory model for the problem – one that is comprehensive, rigorous, and usable – has perhaps been less dramatic. Some models have been proposed, but no clear favorite has yet emerged.

One important open problem is general constant-round non-cryptographic secure distributed computation with polynomial-sized messages. The problem has been solved for functions with log-depth circuits [5], or allowing exponential-length messages [5], or by reducing the resiliency [12], or by assuming the existence of one-way functions [14].

Another open question is the characterization of privately-computed multi-input functions. The two-input privately-computable functions have been completely characterized [8] [73].

A third open question is a more general question of efficiency. Most of the secure distributed computation schemes described in this paper are designed to be applicable to any possible function. For this reason, these schemes are possibly impractical to apply to any actual function that might arise in a setting where secure distributed computation was desired. One example is for distributed databases, where data stored at mutually untrustworthy sites may be needed to answer relational queries. A second example is for signal processing, where model-based conclusions may be sought from sensor data collected by mutually untrustworthy agencies. A third example is multi-criterion decision-making (generalized voting), where individual rankings are kept confidential while somehow merged into an acceptable aggregate ranking. A fourth example is after-hours financial trading systems, where individual bids and offers must be matched fairly while balancing trader privacy with regulatory oversight. It may be necessary to discover special-purpose efficient computation protocols for such specific settings before the beautiful ideas discussed in this paper achieve their full potential.

References

[1] M. Abadi, J. Feigenbaum, and J. Kilian, "On hiding information from an oracle," J. Comput. System Sci. **39** (1989), 21-50.

[2] M. Abadi and J. Feigenbaum, "Secure circuit evaluation: a protocol based on hiding information from an oracle," J. Cryptology **2** (1990), 1-12.

[3] L. Babai and S. Moran, "Arthur-Merlin games: A randomized proof system and a hierarchy of complexity classes," J. Comput. System Sci. **36** (1988), 254-276.

[4] I. Barany and Z. Furedi, "Mental poker with three or more players," Information and Control **59** (1983), 84-93.

[5] J. Bar-Ilan and D. Beaver, "Non-cryptographic fault-tolerant computing in a constant number of rounds of interaction," PODC 1989, 201-209.

[6] D. Barrington, "Bounded-width branching programs recognize exactly those languages in NC^1, J. Comput. System Sci. **38** (1989), 150-164.

[7] R. Bar-Yehuda, B. Chor, and E. Kushilevitz, "Privacy, additional information, and communication," IEEE Structure in Complexity Theory 1990, 55-65.

[8] D. Beaver, "Perfect privacy for two-party protocols," DIMACS Workshop on Distributed Computing and Cryptography, Feigenbaum and Merritt (eds.), AMS, 1990, 65-77.

[9] D. Beaver, "Foundations of secure interactive computing," Crypto 1991, 377-391.

[10] D. Beaver, "Distributed computations tolerating a faulty minority, and multiparty zero-knowledge proof systems," J. Cryptology, **4** (1991), 75-122.

[11] D. Beaver and J. Feigenbaum, "Hiding instances in multioracle queries," STACS 1990, 37-48.

[12] D. Beaver, J. Feigenbaum, J. Kilian, and P. Rogaway, "Security with low communication overhead," Crypto 1990, 62-76.

[13] D. Beaver and S. Goldwasser, "Multiparty computation with faulty majority," IEEE FOCS 1989, 468-473.

[14] D. Beaver, S. Micali, and P. Rogaway, "The round complexity of secure protocols," ACM STOC 1990, 503-513.

[15] M. Bellare, L. Cowen, and S. Goldwasser, "On the structure of secret key exchange protocols," DIMACS Workshop on Distributed Computing and Cryptography, Feigenbaum and Merritt (eds.), AMS, 1990, 79-92.

[16] M. Ben-Or, R. Canetti, and O. Goldreich, "Asynchronous secure computation," ACM STOC 1993, 52–61.

[17] J. Benaloh (Cohen), "Secret sharing homomorphisms: keeping shares of a secret secret," Crypto '86, 251-260.

[18] J. Benaloh (Cohen) and M. Yung, "Distributing the power of a government to enhance to privacy of voters," PODC 1986, 52-62.

[19] M. Ben-Or and R. Cleve, "Computing algebraic formulas using a constant number of registers," ACM STOC 1988, 254-257.

[20] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation," ACM STOC 1988, 1-9.

[21] R. Berger, R. Peralta, and T. Tedrick, "A provably secure oblivious transfer protocol," Eurocrypt 1984, 379-386.

[22] E. Berlekamp, *Algebraic Coding Theory*, Aegean Park Press, Laguna Hills, CA, 1984.

[23] M. Blum, "Three applications of the Oblivious Transfer: Part I: Coin flipping by telephone; Part II: How to exchange secrets; Part III: How to send certified electronic mail," Department of EECS, University of California, Berkeley, CA, 1981.

[24] M. Blum, "Coin flipping by telephone: a protocol for solving impossible problems," IEEE Computer Conference 1982, 133-137.

[25] M. Blum, "How to exchange (secret) keys," ACM Trans. Comput. Sys. **1** (1983), 175-193.

[26] M. Blum, "How to prove a theorem so no one else can claim it," Proc. of the International Congress of Mathematicians, Berkeley, CA, 1986, 1444-1451.

[27] M. Blum, U. Vazirani, and V. Vazirani, "Reducibility among protocols," Crypto 1983, 137-146.

[28] G. Brassard, D. Chaum, and C. Crépeau, "Minimum disclosure proofs of knowledge," J. Comput. System Sci. **37** (1988) 156-189.

[29] G. Brassard, C. Crépeau, and J. Robert, "Information theoretic reductions among disclosure problems," IEEE FOCS 1986, 168-173.

[30] G. Brassard, C. Crépeau, and M. Yung, "Perfectly concealing computationally convincing interactive proofs in constant rounds," Theoretical Computer Science (to appear).

[31] M. Burrows, M. Abadi, and R. Needham, "Authentication: A practical study in belief and action," Proceedings of the Second Conference on Theoretical Aspects of Reasoning about Knowledge," Moshe Vardi (ed.), Morgan Kaufmann, 1988.

[32] D. Chaum, "Untraceable electronic mail, return addresses and digital pseudonyms," CACM **24** (1981), 84-88.

[33] D. Chaum, "Security without identification: Transaction systems to make Big Brother obsolete," CACM **28** (1985), 1030-1040.

[34] D. Chaum, "The spymasters double-agent problem: multiparty computations secure unconditionally from minorities and cryptographically from majorities," Crypto 1989, 591-601.

[35] D. Chaum, C. Crépeau, and I. Damgård, "Multiparty unconditionally secure protocols," ACM STOC 1988, 11-19.

[36] D. Chaum, I. Damgård, and J. van de Graaf, "Multiparty computations ensuring privacy of each party's input and correctness of the result," Crypto 1987, 87-119.

[37] B. Chor, M. Gereb-Graus, and E. Kushilevitz, "Private computations over the integers," IEEE FOCS 1990, 335-344.

[38] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch, "Verifiable secret sharing and achieving simultaneity in the presence of faults," IEEE FOCS 1985, 383-395.

[39] B. Chor and E. Kushilevitz, "A zero-one law for boolean privacy," ACM STOC 1989, 62-72.

[40] R. Cleve, "Limits on the security of coin flips when half the processors are faulty," ACM STOC 1986, 364-369.

[41] R. Cleve, "Controlled gradual disclosure schemes for random bits and their applications," Crypto 1989, 573-588

[42] J. (Benaloh) Cohen and M. Fisher, "A robust and verifiable cryptographically secure election scheme," IEEE FOCS 1985, 372-382.

[43] D. Coppersmith, "Cheating at mental poker," Crypto 1985, 104-107.

[44] C. Crépeau, "A secure poker protocol that minimizes the effect of player coalitions," Crypto 1985, 73-86.

[45] C. Crépeau, "A zero-knowledge poker protocol that achieves confidentiality of the players' strategy, or How to achieve an electronic poker face," Crypto 1986, 239-247.

[46] C. Crépeau, "Equivalence between two flavours of Oblivious Transfer," Crypto 87, 350-354.

[47] C. Crépeau and J. Kilian, "Achieving oblivious transfer using weakened security assumptions," IEEE FOCS 1988, 42-52.

[48] D. Dolev, C. Dwork, O. Waarts, and M. Yung, "Perfectly secure message transmission," JACM 40 (1993), 17–47.

[49] D. Dolev and A. Yao, "On the security of public key protocols," ACM FOCS 1981, 350-357.

[50] S. Even, O. Goldreich, and A. Lempel, "A randomized protocol for signing contracts," CACM **28** (1985), 637-647.

[51] U. Feige, A. Fiat, and A. Shamir, "Zero-Knowledge Proofs of Identity," J. Cryptology, Vol.1, No. 2, 1988, pp. 77-94.

[52] P. Feldman and S. Micali, "Optimal algorithms for Byzantine agreement," ACM STOC 1988, 148-161.

[53] S. Fortune and M. Merritt, "Poker protocols," Crypto 1984, 454-464.

[54] M. Franklin and M. Yung, "Communication complexity of secure computing," ACM STOC 1992, 699-710.

[55] Z. Galil, S. Haber, and M. Yung, "Cryptographic computation: secure fault-tolerant protocols and the public-key model," Crypto 1987, 135-155.

[56] O. Goldreich, S. Micali, and A. Wigderson, "Proofs that yield nothing but their validity and a methodology of cryptographic protocol design," IEEE FOCS 1986, 174-187.

[57] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game," ACM STOC 1987, 218-229.

[58] O. Goldreich and R. Vainish, "How to solve any protocol problem – an efficiency improvement," Crypto 1987, 73-86.

[59] S. Goldwasser and L. Levin, "Fair computation of general functions in presence of immoral majority," Crypto 1989, 75-84.

[60] S. Goldwasser and S. Micali, "Probabilistic encryption," J. Comput. System Sci. **28** (1984) 270-299.

[61] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems," SIAM J. Comput. **18** (1989), 186-208.

[62] S. Haber, "Multiparty cryptographic computation: techniques and applications," Ph.D. thesis, Columbia University, 1988.

[63] J. Håstad, "Pseudo-random generators under uniform assumptions," ACM STOC 1990, 395-404.

[64] J. Håstad and A. Shamir, "The cryptographic security of truncated linearly related variables," STOC 1985, 356-362.

[65] M. Huang and S. Teng, "Security, verifiability, and universality in distributed computing," J. Algorithms **11** (1990), 492-521.

[66] R. Impagliazzo, L. Levin, and M. Luby, "Pseudorandom number generation from one-way functions," ACM STOC 1989, 12-24.

[67] R. Impagliazzo and M. Luby, "One-way functions are essential for complexity based cryptography," IEEE FOCS 1989, 230-235.

[68] R. Impagliazzo and S. Rudich, "Limits on the provable consequences of one-way permutations," ACM STOC 1989, 44-61.

[69] R. Impagliazzo, and M. Yung, "Direct minimum-knowledge computation," Crypto 1987, 40-51.

[70] J. Kilian, "Founding cryptography on oblivious transfer," ACM STOC 1988, 20-31.

[71] J. Kilian, *Uses of Randomness in Algorithms and Protocols*, ACM Distinguished Dissertation Series, The MIT Press, Cambridge, MA 1990.

[72] J. Kilian, "A general completeness theorem for two-party games," ACM STOC 1991, 553–560.

[73] E. Kushilevitz, "Privacy and communication complexity," IEEE FOCS 1989, 416-421.

[74] L. Lamport, R. Shostak, and M. Pease, "The Byzantine generals problem," ACM Trans. on Programming Lang. and Systems (1982), 382-401.

[75] R. Lipton, "How to cheat at mental poker," proceedings of AMS short course on cryptography, 1981.

[76] M. Luby, S. Micali, and C. Rackoff, "How to simultaneously exchange a secret bit by flipping a symmetrically-biased coin," IEEE FOCS 1984, 11-21.

[77] M. Merritt, "Cryptographic protocols," Ph.D. thesis, Georgia Institute of Technology, 1983.

[78] S. Micali and P. Rogaway, "Secure computation," Crypto 1991, 392-404.

[79] M. Naor, "Bit commitment using pseudo-randomness," Crypto 1989, 128-136.

[80] M. Naor, R. Ostrovsky, R. Venkatesan, and M. Yung, "Perfect zero-knowledge arguments for NP can be based on general complexity assumptions," manuscript, 1991.

[81] R. Ostrovsky, R. Venkatesan, and M. Yung, "Fair games against an all-powerful adversary," Sequences Workshop, Positano, Italy, July 1991.

[82] R. Ostrovsky and M. Yung, "On necessary conditions for secure distributed computing," DIMACS Workshop on Distributed Computing and Cryptography, Feigenbaum and Merritt (eds.), AMS, 1990, 229-234.

[83] R. Ostrovsky and M. Yung, "Robust computation in the presence of mobile viruses," ACM PODC, 1991, 51-59.

[84] M. Rabin, "How to exchange secrets by oblivious transfer," Tech. Memo TR-81, Aiken Computation Laboratory, Harvard University, 1981.

[85] T. Rabin, "Robust sharing of secrets when the dealer is honest or cheating," M.Sc. Thesis, Hebrew University, 1988.

[86] T. Rabin and M. Ben-Or, "Verifiable secret sharing and multiparty protocols with honest majority," ACM STOC 1989, 73-85.

[87] R. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public key cryptosystems," CACM **21** (1978), 120-126.

[88] A. Shamir, "How to share a secret," CACM **22** (1979), 612-613.

[89] A. Shamir, R. Rivest, and L. Adleman, "Mental poker," Technical Report MIT/LCS/TR-125, M.I.T., 1979.

[90] G. Simmons, "Geometric shared secret and/or shared control schemes," Crypto 1990, 216-241.

[91] M. Tompa and H. Woll, "Random self-reducibility and zero knowledge interactive proofs of possession of information," IEEE FOCS 1987, 472-482.

[92] U. Vazirani and V. Vazirani, "Trapdoor pseudo-random number generators, with applications to protocol design," IEEE FOCS 1983, 23-30.

[93] A. Yao, "Some complexity questions related to distributive computing," ACM STOC 1979, 209-213.

[94] A. Yao, "Protocols for secure computations," IEEE FOCS 1982, 160-164.

[95] A. Yao, "How to generate and exchange secrets," IEEE FOCS 1986, 162-167.

[96] M. Yung, "Cryptoprotocols: subscription to a public key, the secret blocking and the multi-player mental poker game," Crypto 1984, 439-453.