

Complexity and Security of Distributed Protocols

Matthew Keith Franklin

Submitted in partial fulfillment of the
requirements for the degree
of Doctor of Philosophy
in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

1993

© 1993

Matthew Keith Franklin

All Rights Reserved

Complexity and Security of Distributed Protocols

Matthew Keith Franklin

ABSTRACT

This thesis addresses the topic of secure distributed computation, a general and powerful tool for balancing cooperation and mistrust among independent agents. We study many related models, which differ as to the allowable communication among agents, the ways in which agents may misbehave, and the complexity (cryptographic) assumptions that are made. We present new protocols, both for general secure computation (i.e., of any function over a finite domain) and for specific tasks (e.g., electronic money). We investigate fundamental relationships among security needs and various resource requirements, with an emphasis on communication complexity. A number of mathematical methods are employed for our investigations, including algebraic, graph-theoretic, and cryptographic techniques.

Table of Contents

1. Introduction	5
2. Survey of Secure Distributed Computing	8
3. Communication Complexity of Secure Computation	50
4. Eavesdropping Games	66
5. Joint Encryption and Message-Efficient Secure Computation	90
6. Off-Line Electronic Cash	100
7. Conclusion	118
8. Bibliography	119

Acknowledgments

I am happy to begin by thanking my official advisor, Zvi Galil, and my unofficial advisor, Moti Yung. Zvi gave me a home in his Theory Group. He provided a great environment for doing research, inspired me by his example, and guided me with his advice and encouragement. Moti has been my teacher. From my first weeks at Columbia through to the last revisions of this thesis, he has generously shared his time and knowledge, and patiently shaped me into a scientist. I owe a great debt to my advisors, which I suspect I can only repay by helping others in turn as they have helped me.

I thank Thanasis Tsantilas, Henryk Wozniakowski, Jonathan Gross, Joe Traub, Stuart Haber, Joan Feigenbaum, and Terry Boulton for their support as advisors and as members of my various Committees (Area Paper, Thesis Proposal, and Thesis Defense). With their suggestions and support, they have smoothed my passage through the Ph.D. program.

I thank Renate Valencia, Rosemary Addarich, Mel Francis, Germaine Levesque, and all the other Staff members for countless favors as I navigated bumpily through the Columbia Bureaucracy.

Many others have helped me during my years at Columbia. Pino Italiano was a bountiful source of food, jokes, advice and assistance (and only the jokes were ever less than excellent). I have also enjoyed time spent with the other Galil students – both senior (Raffaele Giancarlo, Kunsoo Park, Dany Breslauer, Xiangdong Yu) and junior (Alain Mayer, Sabah Albinali) – who crossed my path. My officemates Bill Schilit, Hasanat Dewan, Kunsoo Park, Roberto Grossi, and Roberto DePrisco deserve special mention for making the days go by pleasantly. Philip Chan helped get me oriented, George Heineman helped me clear the quals, Bulent Yener helped me keep my perspective, and my “cubicle-mates” on the 11th Floor helped me become a student again. The list of friends and colleagues who made my years here memorable is a long one; I’m sorry I cannot mention them all.

I thank all those who inspired me intellectually before I got to Columbia, and I would like to single out a few of them here. Gilles Brassard hooked me on Cryptology at U.C. Berkeley. The professors of the Math Department at Pomona College gave me an education that has been fundamental and invaluable; special thanks to my undergraduate advisor Sandy Grabiner. Armen Gabrielian showed me how to attack a research topic with imagination, ingenuity, and perseverance; he also taught me how to collaborate, for which I am grateful.

Much of my research has been supported by an AT&T Bell Laboratories Ph.D. Scholarship, which has allowed me to ignore funding worries and concentrate on my work. Summer internships have provided supplemental funding and invaluable experience, thanks to Joan Feigenbaum and Michael Merritt at Bell Labs, Stuart Haber and Peter Winkler at Bellcore, and Moti Yung and Shay Kutten at IBM Research.

I owe my largest debt to my family. Nancy never lost faith in my ability. She got me back to school, inspired me, nurtured me, and encouraged me. She helped me over every hurdle to this degree, and the achievement is as much hers as it is mine. My family in New York has always been there for me when I needed them, and opened their homes to me. My family in California has given me all their love and support over the years. I thank them for their unwavering commitment through good times and hard times. In particular, I wish to express my deep gratitude to my parents for all that they have done for me; I dedicate this thesis to them.

Chapter 1

Introduction

This thesis addresses the topic of secure distributed computation, a general and powerful tool for balancing cooperation and mistrust among independent agents. In a secure distributed computation, a group of parties are mutually untrustworthy, and yet must somehow cooperate to perform some useful task. Typically, each party holds some part of the input to a publicly known function, and the parties attempt to find the appropriate output of the function without compromising the secrecy of the input. A secret ballot election is one simple example of such a task, in which the public function is addition, and each piece of the input is the marked ballot of an individual voter. A sealed-bid auction is another example, in which the public function takes a sequence of individual bids and returns the index of the maximum bid.

When studying secure distributed computation, many aspects of the model are important, including the particular function or functions that can be computed, the number of parties that can participate, the manner of communication among parties, the way in which parties may misbehave, the desired level of security for the parties, and the complexity theoretic (cryptographic) assumptions that are made. Various measures of performance are also important, including the number of faulty parties that can be tolerated, the communication complexity of the protocol (number of rounds of communication, or number of bits transmitted), the amount of randomness needed, and the amount of local computation performed by the parties.

This thesis investigates the inherent tradeoffs among these various assumptions and measures. Tradeoffs that are considered include communication complexity versus fault tolerance (Chapter 3), communication complexity versus level of security (Chapter 3), and number of faults tolerated versus manner of communication (Chapter 4). The increased understanding of secure distributed computation that comes from an analysis of the interrelationship of its relevant parameters is one of the main contributions of the thesis.

Another main contribution of this thesis is to work towards the practical applicability of secure distributed computation. Although theoretical solutions exist for many problems, the communication costs (round and message complexity) of most of these general approaches are prohibitive. We demonstrate many ways to reduce these costs, including offering decreased protection versus strong fault models (Chapter 3), performing many secure distributed computations in parallel (Chapter 3), developing special purpose cryptographic tools (Chapter 5), and designing customized protocols for specific computations (Chapters 5 and 6). If communication costs can be controlled, then the beautiful ideas of many researchers may be applicable to real-world situations that must combine

collaborative effort with security needs.

The reader will get a feeling for some of these ideas in Chapter 2, which gives an extensive survey of previous work in secure distributed computation.

Chapter 3 initiates the study of the communication complexity of secure distributed computation, applying algebraic techniques. In this chapter, we focus on the “unconditional” (or “non-cryptographic”) “complete network” setting. An untappable channel is assumed to connect each pair of parties, such that communication across it cannot be overheard by any other party, while no further complexity theoretic assumptions are made. We answer questions about amortized message complexity in this setting, i.e., whether the average number of bits of communication per computation can be reduced by computing the same function many times in parallel. Interestingly, amortized improvement is possible for one function (addition) in a weak failure model (passive, “gossiping” faults), while impossible for the same function in a slightly stronger failure model (crash faults). These amortization results require that non-trivial lower bounds be found for message complexity of secure computation. Our upper bounds for message complexity take advantage of a general technique for parallelizing many types of secure protocols in a way that trades off message complexity and fault tolerance. In this chapter, we also demonstrate how to reduce the message complexity for one of the strongest adversaries (Byzantine failures), if the detection of faults is adequate protection for the honest parties. The error detecting protocol makes use of our general parallelization technique in a novel way.

Chapter 4 is also concerned with the unconditional setting, but here the network of untappable channels is not necessarily complete. A graph-theoretic approach is initiated by this investigation. We apply combinatorial techniques to study one particular class of adversary: passive mobile eavesdroppers that learn messages and memory contents by moving among the nodes of the network. We characterize the feasibility of maintaining security for two fundamental tasks: sending a secret message across the network, and storing a secret value in the network. For secure message storage, feasibility is quite sensitive to the constraints that are placed on the movement of eavesdroppers, as well as to the topology of the communication network. In fact, we establish a close connection between the secure storage problem and a graph searching problem – unrelated to security – that has been much studied for the past twenty years. For secure message transmission, on the other hand, feasibility is completely insensitive to the constraints placed on the eavesdroppers, and depends only on their number in relation to connectivity properties of the communication network. In addition to characterizing these feasibility conditions, we also find the computational complexity of determining feasibility for a number of important cases.

Chapter 5 considers communication complexity of secure distributed computation in the “cryptographic” setting, in which some specific complexity theoretic assumption is made. We show that an improvement (linear in the number of parties) in the number of bits communicated is possible under a particular cryptographic assumption. This improvement is achieved by drawing a connection to the previously unrelated topic of group-oriented cryptography, in which the public-key model is extended from individuals to sets of participants. The communication decrease for secure computation follows from the construction of a new group-oriented encryption scheme, based on concrete number-theoretic assumptions, which is of independent interest. We also demonstrate how decreased communication for secure distributed computation may be possible by designing specific protocols for specific computational problems.

Chapter 6 discusses new results in the area of electronic money in the cryptographic setting.

Rather than a single secure distributed computation, electronic money is a collection of interrelated protocols that manage the flow of funds among an economy of banks, shopkeepers, and consumers. We give new methods and mechanisms for the tricky problem of “off-line coins”, for which the identity of a consumer must remain hidden if any coin is spent only once, while the identity must be revealed if the coin is spent more than once. We also give new methods for the related problem of “blind signature,” in which one party must be able to guarantee some property of the message it is signing without being able to recognize its signature afterwards.

Most of the results in this thesis have appeared previously in preliminary forms as extended abstracts at various conferences. The work from Chapter 3 is discussed in “Communication complexity of secure computation,” which is joint work with Moti Yung, and appears in the 1992 STOC conference proceedings [76]. The results from Chapter 4 are covered in “Eavesdropping games: a graph-theoretic approach to privacy in distributed systems,” which is joint work with Zvi Galil and Moti Yung, and appears in the 1993 FOCS conference proceedings [74]. The results in Chapter 5 are discussed in “Joint encryption and message efficient secure computation,” which is joint work with Stuart Haber, and appears in the 1993 Crypto conference proceedings [75]. The research in Chapter 6 is covered in “Secure and efficient off-line digital cash,” which is joint work with Moti Yung, and appears in the 1993 ICALP conference proceedings [77].

Chapter 2

A Survey of Secure Distributed Computing

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 chapter, 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 chapter is organized.

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 [132] [134] 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 [26] is a two-party protocol that arrives at a common random bit; this was one of the earliest cryptographic protocols. Mental poker [133] [105] [51] [85] [141] [73] [52] [53] 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 [26] [23] [34] 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 [28] [31] [107] [89] [140] 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 [38] [39] is a collection of protocols (e.g, withdrawal, purchase, deposit) that implement payment schemes without any physical requirements. Secret-ballot election schemes [50] [19] [91] 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 [62] [111], proving security by relating actual protocols homomorphically to theoretically perfect protocols. A second approach is logical [37], 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] [13] 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 [81], and later improved by [95]. More will be said about zero-knowledge protocols in Section 2.1.5.

Organization of the Chapter

The next section of this chapter gives short descriptions of cryptographic and other primitives that are used in the upcoming protocols. Section 2.2 presents one model for secure distributed computation, as well as pointing out where “competing” models differ. Two-party cryptographic protocols are presented in Section 2.3, multi-party cryptographic protocols in Section 2.4, and multi-party non-cryptographic (sometimes called “unconditional”) protocols in Section 2.5. Section 2.6 presents some of the complexity results and lower bounds that are known for these protocols, and

Section 2.7 closes the chapter with some discussion and open questions.

2.1 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.

2.1.1 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 $g^i = a \pmod p$, $0 \leq i < p$. 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 $\sigma_1, \dots, \sigma_k$ such that (1) if r is chosen uniformly at random from $\text{dom}(f)$ then each $\sigma_i(x, r)$ is uniformly distributed over $\text{dom}(f)$ for every $x \in \text{dom}(f)$, and (2) $f(x) = \phi(x, r, f(\sigma_1(x, r)), \dots, f(\sigma_k(x, r)))$ for all $x, r \in \text{dom}(f)$. The function QR_n is random self-reducible over Z_n^* (for any n), since (1) $xr \pmod n$ is a uniformly distributed element of Z_n^* whenever r is chosen at random from Z_n^* and (2) $QR_n(x) = QR_n(xr \pmod n) \oplus QR_n(r)$. The function $DLP_{p,g}(a)$ is random self-reducible over Z_p^* since (1) $ag^r \pmod p$ is uniformly distributed for random $r \in Z_p^*$ and (2) $DLP_{p,g}(a) = DLP_{p,g}(ag^r \pmod p) - r$.

2.1.2 Indistinguishable Probability Distributions

The indistinguishability of probability distributions, due to Yao [139] and Goldwasser and Micali [85], 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.

2.1.3 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 [129]: $E(m) = m^e \bmod n$, 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” [85], 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.

2.1.4 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 [27] 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 [114] shows a general construction for basing bit commitment on any one-way function, based on earlier reductions [92] [88].

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” [33] [35] [115] allows the receiver to be unbounded. In this case, unreadability requires that the two probability distributions be (almost) identical.

2.1.5 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 [86] and Babai [4]. 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” [86] 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 [29]. 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 [81] 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 [69] [136], 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.

2.1.6 Secret Sharing and Verifiable Secret Sharing

Underlying many recent results in secure distributed computing is a secret sharing scheme due to Shamir [132]. 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 i th 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 [46], 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 [22] 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 [127] that can tolerate any minority of faulty players with an exponentially small probability of error.

2.1.7 Instance Hiding

Instance hiding schemes ([1], [13]) 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 [13] (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 $g^j = ag^i \pmod p$, 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] [14].

2.2 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 [126] and Byzantine Agreement [103]. Some of the difficulties of formalizing security for protocols are discussed, along with some recent attempts to overcome these difficulties.

2.2.1 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 communication 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 i th 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 $f(x_1, \dots, x_n)$ (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 $f(s_1, \dots, s_n)$ 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 [6] and Beaver, Feigenbaum, Kilian, and Rogaway [14]).

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 [128] [61].

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 [78] are active adversaries that are limited to withholding outgoing messages from a faulty processor. Independent adversaries [61] 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 [122] [61]), there is either a single static passive adversary or a single static active adversary.

2.2.2 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 [126] (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 [34] [54] [55]. One important alternative is called “1-2 Oblivious Transfer” [65] (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 [96] [15] [84], as will be discussed in later sections. To give a simpler reduction now, the following scheme, due to Crépeau [54], achieves bit commitment through oblivious transfer:

1. Player A chooses random b_1, \dots, b_n such that $b_1 \oplus \dots \oplus b_n = b$.
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 $b = b_1 \oplus \dots \oplus b_n$ 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 [103], 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 is not to conceal the inputs, and although channels are not private in the original model¹ [103]). 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 $f(b_1, \dots, b_n)$ on its output tape, where $f(b_1, \dots, b_n) = b_1$ if $b_1 = \dots = b_n$ 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.

2.2.3 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.

¹A fast probabilistic protocol for Byzantine Agreement in a network of private channels is due to Feldman and Micali [70]; 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 [103] [70], 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 $n > 3t$.

2.2.4 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 [84], will not be discussed.

One proposal to put this area on a more formal foundation is due to Beaver [11]. 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 [113], 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

²This 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.

2.3 Two-Party Cryptographic Secure Distributed Computation

2.3.1 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 [22] 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 [139]. 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.

2.3.2 Protocols

The first general two-party secure computation protocol result is due to Yao [140], 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 [82] 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, \dots, 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 $r_1, r'_1, \dots, r_m, r'_m, r_x, r'_x, r_y, r'_y$.
2. B chooses random bijections $\phi_1, \dots, \phi_m, \phi_x, \phi_y$ 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}, \dots, 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. $p_1 = \langle E(m_{i1}, r_{left}), E(m'_{i1}, r_{right}) \rangle$
 - ii. $p_2 = \langle E(m_{i2}, r'_{left}), E(m'_{i2}, r_{right}) \rangle$
 - iii. $p_3 = \langle E(m_{i3}, r_{left}), E(m'_{i3}, r'_{right}) \rangle$
 - iv. $p_4 = \langle E(m_{i4}, r'_{left}), E(m'_{i4}, r'_{right}) \rangle$
 - (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. $B \rightarrow A$: $S_1, S_2, \dots, S_m, \phi_y^{-1}(b), \phi_m = \{ \langle r_m, \phi_m(r_m) \rangle, \langle r'_m, \phi_m(r'_m) \rangle \}$.
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, \dots, S_m . Originally label all of the AND and NOT gates g_1, g_2, \dots, 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 occurrence 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 occurrence 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, \dots, 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 [65]).

Galil, Haber, and Yung [78] 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 [92] [88]) that returns the output decryption key (yet another seed for the generator).

Kilian [96] 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 [54]. 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 [7]) 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 [42] 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 $a^{-1} x y$ 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, \dots, 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. $B \rightarrow A : y_1, y_2, \dots, y_m$.
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.

2.4 Multi-Party Cryptographic Secure Computation

2.4.1 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 [82]: 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 [78], 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 2.1: Summary of Goldreich, Micali, and Wigderson 1987 (passive adversary)

Some protocols (beginning with the results of Goldreich, Micali, and Wigderson [82]) 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 [81]. 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 subsections. 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).

2.4.2 Multi-Party Cryptographic Protocols versus Passive Adversaries

A two-party protocol for any two-input computation, due to Goldreich, Micali, and Wigderson [82], 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 [7], 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 2.2: Summary of Galil, Haber, and Yung 1987

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

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., $\sigma \circ \sigma' = (\sigma_1 \circ \dots \circ \sigma_n) \circ (\sigma'_1 \circ \dots \circ \sigma'_n) \neq (\sigma_1 \circ \sigma'_1) \circ \dots \circ (\sigma_n \circ \sigma'_n)$. 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 [78] 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 functions 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}, \dots, 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 with 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 [82] “and-xoring,” i.e., $f((x, y), z) = x \oplus (y \wedge z)$ for [78]. 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 [83] 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).

2.4.3 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 2.3: Summary of Goldreich, Micali, and Wigderson 1987 (active adversary)

Goldreich, Micali, and Wigderson [82] 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 [78] 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 2.4: 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 [42] 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, \dots, 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 $S = \{ \langle 0, 0, 0, \emptyset \rangle, \langle 0, 1, 0, \emptyset \rangle, \langle 1, 0, 0, \emptyset \rangle, \langle 1, 1, 1, \emptyset \rangle \}$ 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 $S = \{ \langle 0, 1, \emptyset \rangle, \langle 1, 0, \emptyset \rangle \}$ 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 i th 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 $c_{x_1}, \dots, c_{x_n}, c_1, \dots, c_m$, 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 $S = \{ \langle a_{11}, a_{12}, a_{13}, R_1 \rangle, \dots, \langle a_{41}, a_{42}, a_{43}, R_4 \rangle \}$.
 - 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 $r_j = r_{j1} \oplus r_{j2} \oplus \dots \oplus r_{ji}$.
 - G. Replace each $(i-1)$ -transform entry a_{j3} with $a_{j3} \oplus r_j$, $1 \leq j \leq 4$.
 - H. *Modify* the k th 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 $S = \{ \langle a_{11}, a_{12}, R_1 \rangle, \langle a_{21}, a_{22}, R_2 \rangle \}$.
 - 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 $r_j = r_{j1} \oplus r_{j2} \oplus \dots \oplus r_{ji}$.
 - F. Replace each $(i-1)$ -transform entry a_{j2} with $a_{j2} \oplus r_j$, $1 \leq j \leq 2$.
 - G. *Modify* the k th 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, \dots, 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 $S = \{ \langle a_{11}, a_{12}, a_{13}, R_1 \rangle, \dots, \langle a_{41}, a_{42}, a_{43}, R_4 \rangle \}$ [$S = \{ \langle a_{11}, a_{12}, R_1 \rangle, \langle a_{21}, a_{22}, R_2 \rangle \}$].
 - 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 [40], 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.

2.4.4 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 [26]. 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 [28], and gave an incorrect solution (generalized and corrected by Yao [139]) 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 [107]: 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 [137] investigated a similar but somewhat weaker notion of exchanging a secret for a valid receipt.

Yao [140] 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 [78] 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 [28], and following Yao [140]) 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 [15] present a different way of achieving fairness for boolean functions (similar to Luby *et al.* [107]), 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:	$\text{poly}(C, n)$
round complexity:	$O(D)$

Table 2.5: Summary of Beaver and Goldwasser 1989

functions is known to follow from the feasibility of oblivious transfer [17].

Goldwasser and Levin [84] 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 [49] 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.

2.5 Non-Cryptographic Secure Distributed Computation Protocols

2.5.1 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 [5], 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 [132]. 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” [82]. 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 [18]) 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 [22]. For greater resilience, new techniques of verifiable secret sharing [127] 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:	$O(n^2 C \log F)$
round complexity:	$O(D)$

Table 2.6: Summary of Ben-Or, Goldwasser, and Wigderson 1988 (passive adversary)

2.5.2 Non-cryptographic Protocols versus Passive Adversaries

The first two papers suggesting general non-cryptographic distributed computation are by Ben-Or, Goldwasser and Wigderson [22], and by Chaum, Crépeau, and Damgård [41]. 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 [22] 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 $\lambda(s_1, \dots, s_n)$ 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:	$O(n^2C \log F)$
roundcomplexity:	$O(D)$

Table 2.7: 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 i th component of $(v_1v_2 \cdots v_n)B^{-1}Chop_tB$, 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 $(w_{1i}; w_{2i} \cdots w_{ni}) = (v_{1i}v_{2i} \cdots v_{ni})B^{-1}Chop_tB$
 - (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 $(1, w_{i1}), (2, w_{i2}), \dots, (n, w_{in})$. 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 $B^{-1}Chop_tB$. 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:	$O(n^5 C \log F)$
round complexity:	$O(D)$

Table 2.8: 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.

2.5.3 Non-cryptographic Protocols versus Active Adversaries

In this subsection, we will cover four non-cryptographic protocols for distributed computation versus an active adversary. The first two [22] [41] achieve t -resilience whenever there are $n > 3t$ players, assuming a complete private network of channels. The last two [128] [12] 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 [22] 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 receive values at some arbitrary α_i (e.g., $\alpha_i = i$), this protocol requires that each player receive values at a specific n th 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 n th 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 [24]. 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 [12] 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 [41] 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 [78] 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 2.9: 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 [22] [41] 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 [128], and by Beaver [12], both building on ideas from T. Rabin [127].

At the heart of the protocol is a verifiable secret sharing scheme, due to T. Rabin [127], 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:	unconditional
Error Prob:	zero
bit complexity:	$\text{poly}(C, n, F)$
round complexity:	$O(1)$

Table 2.10: 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 $\bar{p}(x)$ with the same coefficients for all of its terms. Each output share is in fact a specific linear combination of the input shares: $\bar{p}(\alpha_i) = \sum_{j=i}^n \bar{L}_j(\alpha_i)p(\alpha_j)$, where $\bar{L}_j(x)$ is the (publicly computable) degree t truncation of the Lagrange polynomial which is one at α_j and zero at $\alpha_i, i \neq j$.

Re-randomization is straightforward, due to Beaver's observation [12] 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.

2.5.4 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 [6] 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 [21], 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:	$\text{poly}(n, \log F)$
round complexity:	$O(1)$

Table 2.11: 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 [96]), this long product of matrices can be securely computed using only two secret matrix multiplications:

$$\begin{aligned}
 Y &= M_1 M_2 \cdots M_N = R_0 (R_0^{-1} M_1 R_1) (R_1^{-1} M_2 R_2) \cdots (R_{N-1}^{-1} M_N R_N) R_N^{-1} \\
 &= R_0 Z R_N^{-1}.
 \end{aligned}$$

Thus, if the players jointly create and share random invertible matrices R_0, \dots, 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 [14] 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 [6]. 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 [16] 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 [140] 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 2.12: 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 2.13: 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 [92] and Håstad [88].

2.5.5 Hybrid Computation Protocols

Chaum [40] 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 $f(s_1, \dots, s_n)$ into an $(n + 1)$ -ary function evaluation $f^*(s_1 \oplus r_1, \dots, s_n \oplus r_n, r_1 \circ \dots \circ r_n)$, 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 [42] (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., [41]) 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.

2.5.6 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 [128] 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 [61] 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.

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

Ostrovsky and Yung [122] 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.

2.5.8 Non-Cryptographic Asynchronous Computation

Ben-Or, Canetti and Goldreich [20] 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.

2.6 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.

2.6.1 Reductions Among Primitives

Kilian [97] 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 [98], 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 [120] 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 [96] 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 [93] 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 [94], 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 [121] demonstrate the necessity of a complete network of private channels for implementing private multiparty computation among computationally unbounded players (sufficiency was shown by [22] and [41]). 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.

2.6.2 Necessary Conditions and Resources

The first complexity results in this field are two impossibility results due to Yao [139], 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 [22] 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 [103].

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 [55] 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 [48] 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 [101] and by Beaver [10]. 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 [8] 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 2.1: 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 2.2: 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 chapter assume that the computation is performed over some finite field. Chor, Gera-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 [138]. 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 [61] 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.

2.7 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 [6], or allowing exponential-length messages [6], or by reducing the resiliency [14], or by assuming the existence of one-way functions [16].

Another open question is the characterization of privately-computed multi-input functions. The two-input privately-computable functions have been completely characterized [10] [101].

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 chapter achieve their full potential.

Chapter 3

Communication Complexity of Secure Computation

In this chapter, we initiate the investigation of the communication complexity of unconditionally secure multi-party computation, and its relation with various fault-tolerance models. We present upper and lower bounds on communication, as well as tradeoffs among resources.

First, we consider the “direct sum problem” for communication complexity of perfectly secure protocols: Can the communication complexity of securely computing a single function $f : F^n \rightarrow F$ at k sets of inputs be smaller if all are computed simultaneously than if each is computed individually? We show that the answer depends on the failure model. A factor of $O(\frac{n}{\log n})$ can be gained in the privacy model (where processors are curious but correct); specifically, when f is n -ary addition (mod 2), we show a lower bound of $\Omega(n^2)$ bits for computing f once and an upper bound of $O(n^2 \log n)$ for computing f $O(n)$ times simultaneously. No gain is possible in a slightly stronger fault model (fail-stop mode); specifically, when f is n -ary addition over $GF(q)$, we show an exact bound of $\Theta(kn^2 \log q)$ for computing f at k sets of inputs simultaneously (for any $k \geq 1$).

However, if one is willing to pay an additive cost in fault tolerance (from t to $t - k + 1$), then a variety of known non-cryptographic protocols (including “provably unparallelizable” protocols from above!) can be systematically compiled to compute one function at k sets of inputs *with no increase in communication complexity*. Our compilation technique is based on a new compression idea of polynomial-based multi-secret sharing.

Lastly, we show how to compile private protocols into error-detecting protocols at a big savings of a factor of $O(n^3)$ (up to a log factor) over the best known error-correcting protocols. This is a new notion of fault-tolerant protocols, and is especially useful when malicious behavior is infrequent, since error-detection implies error-correction in this case.

3.1 Introduction

Secure computation protocols allow the cooperative evaluation of an arithmetic circuit by a group of processors, where each processor initially knows one input to the circuit. The protocol must not leak information about other processors’ secret inputs, even if some group of processors misbehaves. Misbehavior may be passive (e.g., pooling legally obtained information), or active (e.g., violating the protocol in an arbitrary coordinated attack). The basic problem of designing protocols for arbitrary

arithmetic circuits was first solved for two processors by Yao [140], and for any number of processors by Goldreich, Micali, and Wigderson [82]. These solutions relied on unproven assumptions: the intractability of factoring, and the existence of trapdoor functions, respectively.

Starting with Ben-Or, Goldwasser, and Wigderson [22], and Chaum, Crépeau, and Damgård [41], more recent protocols have removed the reliance of multi-party protocols on unproven assumptions. These protocols rely instead on the physical assumption that untappable communication channels connect all pairs of processors; sometimes these are called “unconditional” or “non-cryptographic” protocols.

In this chapter, we initiate the investigation of the communication complexity of unconditionally secure multi-party fault-tolerant protocols (where by fault-tolerant we mean able to withstand attack by a *constant fraction* of faulty processors¹). Since communication is a crucial resource, it is both natural and important to understand its intrinsic limits by showing lower bounds, and to reduce it by presenting improved upper bounds. Previously, communication complexity for secure computation was studied only for the privacy model and only for the two-party case [101] [8] (see also [119] for results in the related model of cooperative two-party computation versus an eavesdropper). Recently, and independently, Kushilevitz [102] has studied the communication complexity of multi-party private addition (see our Lemma 3.2).

One basic question addressed by this chapter is the “direct sum problem” for the communication complexity of secure computation (recently addressed in the “non-secure” setting by Feder, Kushilevitz, and Naor [68]), i.e., when can parallelization reduce the amortized complexity of secure computation? We show that the answer is “sometimes,” “sometimes not,” and “almost always,” depending (of course) on security setting and fault-tolerance tradeoffs. Lower bound arguments allow us to prove that in one security setting (privacy only, assuming processors are curious but otherwise honest), parallelization can reduce amortized bit complexity, while in a somewhat stronger setting (unstoppability, assuming that processor may stop-fail and requiring the protocol to nevertheless continue) it cannot. If one is willing to pay a small price in fault-tolerance, however, we show a systematic technique for parallelizing many secure computation protocols at no increase in total bit complexity.

A related question addressed by this chapter is how to protect against a very strong adversary without a big communication penalty. Existing protocols offer “resilience,” which gives the honest processors the correct output despite arbitrary coordinated tampering by a subset of faulty processors (analogous to error correcting codes). When malicious behavior is infrequent, however, it suffices to notify the honest processors that tampering has occurred (analogous to error detecting codes). Once notified, the honest processors may repeat the computation (possibly switching to a resilient protocol) to obtain the correct output. We call this more modest protection “detectability,” and develop techniques to achieve it at a significant decrease in bit complexity over resilience.

3.1.1 Organization of the Chapter

In Section 3.2, we present the basic model for secure computation and the basic fault models. In particular, we offer two security notions new in the setting of secure computation: *unstoppability* (to withstand fail-stop mode of failure) and *detectability* (to detect arbitrary deviation from the

¹For $O(\log n)$ faults only, a protocol due to Beaver, Feigenbaum, Kilian, and Rogaway [14] has bit complexity independent of circuit size (dependent on the input size and n); no known protocols tolerating a constant fraction of faulty players have this property.

specified protocol). These requirements are both in between the known “privacy” and “resilience” modes.

In Section 3.3, we describe our general technique for parallelizing non-cryptographic computation protocols, at a small cost in fault-tolerance. Our technique replaces polynomial-based (single) secret sharing with a technique allowing multiple secrets to be hidden in a single polynomial. We illustrate the “multi-shares” technique on a protocol for general computation versus a passive adversary. The technique actually applies to all of the protocols for secure computations which use polynomial-based threshold schemes and applies to all fault-tolerance models.

In Section 3.4, we prove matching lower and upper bounds to answer two direct sum problems for communication complexity of secure protocols. We show that, versus a private adversary controlling $\lfloor \frac{n}{2} \rfloor$ of the processors, $\Theta(n)$ computations of the n -ary addition modulo 2 function can be performed simultaneously using $O(n^2 \log n)$ bits of communication, while $\Omega(n^3)$ bits of communication are required if the computations are performed individually. We also show that, versus a somewhat stronger adversary (fail-stop mode) controlling $\lfloor \frac{n}{2} \rfloor$ of the processors, any k computations of the n -ary addition function over a field of size q costs $\Theta(kn^2 \log q)$; thus, for this fault-tolerance model for computing this problem, parallelization cannot help. This is in contrast with the tradeoff of Section 3.3, where paying with fault-tolerance does help reduce the communication.

In Section 3.5, we present a computation protocol that provides communication-efficient detectability. This protocol is compiled from a protocol for the privacy model. Specifically, our protocol performs the computation on exactly two sets of inputs simultaneously: the real secret inputs, and a collection of jointly-created check inputs. We suggest a technique by which the two computations proceed in *lock-step*, in such a way that one computation cannot be influenced without randomly affecting the other. At the end of the computation, the honest processors learn the (supposed) real output, the (supposed) check output, and the check inputs. We prove that if the honest processors accept the real output only when the check computation is correct, then the probability of undetected error can be made arbitrarily small. The bit complexity of this protocol is about the same as the cheapest known fault-tolerant private protocol although it protects against a much stronger adversary.

3.2 Model and Definitions

A group of n “processors” or “players” jointly evaluate an n -ary function over some finite field F with $|F|$ elements. The function is represented as an arithmetic circuit consisting of 2-ary addition gates, 1-ary scalar multiplication gates, and 2-ary multiplication gates. The class of all arithmetic circuits with exactly M multiplication gates is denoted C_M .

Each player is a probabilistic Turing Machine, and each pair of players is connected by a physically secure communication channel (the “non-cryptographic” or “unconditional” setting). Each player begins with a secret input on its private tape. At the end of a computation protocol, each player has on its output tape the value of the function at the secret input values. Unless otherwise stated, all protocols in this chapter are assumed to be synchronous (rounds of communication interleaved with periods of private computation), have *oblivious* message-size (number of bits sent in round i from player j to player k depends only on i , j , and k), and perfect (have zero probability of being insecure or computing with error – sometimes we relax this to be “almost perfect”).

A protocol is t -*private* if no collection of t players, while following the protocol exactly, gains

any additional information about the other $n - t$ secret inputs beyond what is necessarily revealed by knowing their t secret inputs and the output².

A protocol is *t-resilient* if no collection of t players, while possibly violating the protocol in an arbitrary and coordinated manner, either gains any additional information about the other $n - t$ secret inputs or prevents the honest players from learning an appropriate output of the protocol. Here an appropriate output is somewhat difficult to define (see, e.g., Beaver [11] and Micali and Rogaway [113] for more precise definition).

We introduce the new and natural security requirement of *t-unstoppability*. A protocol is *t-unstoppable* if there is a commit point in the protocol such that no collection of t players, while possibly dropping out of the protocol, either gains any additional information about the other $n - t$ secret inputs or prevents the honest players from learning an appropriate output of the protocol. Here an appropriate output would be the value of the function with default values substituted for all players that dropped out before the commit point. This definition captures the fail-stop behavior, which is actually provided by various protocols in the literature.

We also introduce the new and natural security property of *t-detectability*. A protocol is *t-detecting* if no collection of t players, while possibly violating the protocol in an arbitrary and coordinated way, can either learn any additional information about the other $n - t$ secret inputs, or prevent the honest players from detecting (with high probability) that tampering has occurred. While resilience provides us with correcting capabilities, detectability implies only that violations are observed.

3.3 Compilation Technique for Parallelizing Secure Protocols

In this section, we present a systematic technique to parallelize many known secure computation protocols. For any k , $1 \leq k \leq t$, this technique allows a single function to be computed simultaneously at k distinct sets of inputs. The communication complexity remains the same as in the original sequential version, for an amortized savings of a multiplicative factor of k . The number of rounds remains the same, and the information-theoretic level of security is maintained throughout. The *multiplicative* gain in bit complexity comes at the expense of an *additive* factor of k in the number of faulty players that can be tolerated, i.e., from t for the original sequential version to $t - k + 1$ for the parallelized version.

Thus, for many secure computation protocols in the non-cryptographic setting, the following “meta-theorem” applies:

Meta-Theorem 3.1 (*Generic Parallel Compilation*) *A protocol that computes f at a single set of inputs tolerating t faulty players can be systematically modified into a protocol that computes f at $k < t$ sets of inputs tolerating $t - k + 1$ faulty players. The round complexity, bit complexity, and level of security are unchanged.*

Recall that we consider fault-tolerant protocols that can withstand a constant fraction of faulty players (typically $n \geq 2t + 1$ or $n \geq 3t + 1$). For these protocols, the parallel version can compute f at $\Theta(n)$ sets of inputs while still tolerating a constant fraction of faulty players.

²In this and subsequent definitions, the coordinated behavior of faulty players can be assumed to be under the control of a single (probabilistic Turing Machine) adversary.

3.3.1 Multi-Secret Sharing

A “multi-secret sharing scheme” lies at the heart of our general compilation technique for parallelizing secure protocols. In this subsection, we define multi-secret sharing and present our scheme.

A $(c, d; k, n)$ -multi-secret sharing scheme is a protocol between a Dealer and n players with a Distribution Phase and a Recovery Phase, such that (1) the Dealer begins with k secrets s_1, \dots, s_k ; (2) the Dealer sends a message to each player in the Distribution Phase; (3) any subset of at least d players can reconstruct all k secrets in the Recovery Phase from their received messages; and (4) no subset of at most c players can deduce anything, in an information-theoretic sense, about the k secrets from their received messages. The message sent from the Dealer to a player in the Distribution Phase is called a “multi-share” of the k secrets.

Theorem 3.1 *There is a $(c, c + k; k, n)$ -multi-secret sharing scheme to share k elements of a finite field F among n players, $1 \leq c \leq n - k$, where each multi-share is a single element of F .*

Proof : Let s_1, \dots, s_k be the Dealer’s secrets. Assume that $\alpha_1, \dots, \alpha_n$ and e_1, \dots, e_k are pre-selected elements of F that are known to the Dealer and all n players.

A generalization of Shamir’s secret-sharing scheme [132] suffices. In the Distribution Phase, each player i receives the multi-share $p(\alpha_i)$, where $p(x) \in F[x]$ is an otherwise random degree t polynomial such that $p(e_i) = s_i$, $1 \leq i \leq k$. More specifically, $p(x) = q(x) \prod_{i=1}^k (x - e_i) + \sum_{i=1}^k s_i L_i(x)$, where $q(x)$ is a completely random degree $c - 1$ polynomial, and where $L_i(x)$ is the Lagrange polynomial $\frac{\prod_{j \neq i} (x - e_j)}{\prod_{j \neq i} (e_i - e_j)}$.

Any $t + 1$ players can interpolate their multi-shares to recover $p(x)$, and hence recover all k secrets. For any $t - k + 1$ multi-shares, there is a single polynomial that is consistent with those multi-shares and *any* k secrets. \square

When secrets are shared using this protocol, and when the parameters c , d , k , and n are clear from context, we may say that the k secrets have been “multi-shared” at “secret locations” e_1, \dots, e_k , using “share locations” $\alpha_1, \dots, \alpha_n$, and that $p(\alpha_i)$ is the “multi-share” of the secrets received by player i .

3.3.2 Homomorphic Multi-Shares

If m_1, \dots, m_n are multi-shares of s_1, \dots, s_k , then cm_1, \dots, cm_n are multi-shares of cs_1, \dots, cs_k ; scalar multiplication of multi-shared secrets can thus be performed by the individual players by a private operation on each individual multi-share. If m'_1, \dots, m'_n are multi-shares of secrets s'_1, \dots, s'_k , and if the share locations and secret locations are the same for both sets of multi-shares, then $m_1 + m'_1, \dots, m_k + m'_k$ are multi-shares of $s_1 + s'_1, \dots, s_k + s'_k$; addition of multi-shared secrets is thus also a local operation by each individual player. These nice homomorphic properties of multi-shared secrets are summarized in the following lemma.

Lemma 3.1 (*Homomorphic Multi-Shares*) *Any linear combination of multi-shares is a multi-share of the linear combinations of multi-shared secrets, providing that the same secret locations and the same share locations were used for all multi-shares.*

3.3.3 Parallel Private Computation

In this section, we illustrate our parallelization technique by showing how the t -private arithmetic circuit computation protocol due to Ben-Or, Goldwasser, and Wigderson [22] (which will be called the “BGW-priv” protocol), can be parallelized. Multi-secret sharing replaces Shamir’s single-secret sharing scheme, and the algebra is extended to manipulate multi-shares appropriately. A multiplicative factor of k in the amortized communication complexity is gained; an additive factor of k in the number of faulty players tolerated is required. The same ideas can be applied to the private computation protocol due to Chaum, Crépeau, and Damgård [41] (over fields of the proper form).

In BGW-priv, each player shares its secret input using Shamir’s scheme. In our parallelization, each player multi-shares its k inputs using the $(t - k + 1, t + 1; k, n)$ -multi-sharing scheme from the proof of Theorem 3.1. In BGW-priv, linear computations are simple, due to homomorphic properties of shared secrets. By Lemma 3.1, the same homomorphic properties of multi-shares can be exploited for linear computations.

It suffices to show how the BGW-priv subprotocol for 2-ary multiplication can be parallelized. In BGW-priv, if each player multiplies its shares of two secrets, the result is a “pseudo-share” of the product of the secrets, i.e., it is necessary to re-randomize the polynomial and to reduce its degree from $2t$ to t . In our parallelization, if each player multiplies its two multi-shares, the result is a “pseudo-multi-share” of the k products, similarly failing to be a true multi-share because the underlying polynomial is non-random and degree $2t$.

In BGW-priv, re-randomization is achieved by having each player distribute values of an otherwise random degree $2t$ polynomial which is zero at the single secret location, whereupon each player adds all received values to its pseudo-share. This randomizes the underlying polynomial without affecting its value at the single secret location. For our parallelization, it suffices that the polynomial each player distributes be zero at *all* secret locations.

In BGW-priv, the degree reduction is achieved by noting that it is a linear operation. Specifically, if w_i is the non-degree-reduced pseudo-share of player i , and v_i is the degree-reduced share of player i , then $(v_1 \cdots v_n) = (w_1 \cdots w_n)A$, for some publicly known constant matrix A . More specifically, $A = B^{-1}Chop_t B$, where B is the n by n vandermonde matrix whose (i, j) entry is α_j^{i-1} , and where $Chop_t$ is the n by n matrix whose (i, j) entry is 1 if $1 \leq i = j \leq t$ and 0 otherwise. Intuitively, multiplying by B converts the vector of polynomial values into the vector of polynomial coefficients; multiplying by $Chop_t$ zeros the higher order coefficients; and multiplying by B^{-1} converts back from coefficients to values. Thus degree reduction can be achieved by using the known subprotocol for any linear operation: Singly share the pseudo-shares, and perform the appropriate linear computations.

For our parallelization, degree reduction is also a linear operation. If w_i is the non-degree-reduced pseudo-multi-share of player i , and v_i is the degree-reduced multi-share of player i , then $(v_1 \cdots v_n) = (w_1 \cdots w_n)A$ for a constant matrix A . In this case, $A = \sum_{l=1}^k B_{e_l}^{-1} Chop_{t-k+1} B_{e_l} M_{e_l}$, where B_{e_l} is the n by n vandermonde matrix at e_l whose (i, j) entry is $(\alpha_j - e_l)^{i-1}$, $Chop$ is as before, and M_{e_l} is the n by n matrix whose (i, j) entry is $L_i(e_l) = \frac{\prod_{i' \neq i} (\alpha_i - e_{i'})}{\prod_{i' \neq i} (e_l - e_{i'})}$ if $i = j$ and 0 otherwise. Intuitively, multiplying by B_{e_l} converts the vector of polynomial values into the vector of coefficients under the translation $y = x - e_l$; multiplying by $Chop_t$ zeros the high order coefficients while maintaining the value of the polynomial at e_l ; multiplying by $B_{e_l}^{-1}$ reconverts to a vector of

values; and multiplying by M_{e_i} helps “glue” all the pieces together using Lagrange interpolation. This linear computation can also be performed using the known subprotocol for *sequential* linear operations: Singly share the pseudo-multi-shares, and perform the appropriate linear computations.

By making the systematic modifications described above, we derive a protocol whose properties are summarized by the following theorem.

Theorem 3.2 (*Private Computation on Multi-Shares*) For $n \geq 2t + 1$, $t \geq k \geq 1$, there is a $(t - k + 1)$ -private protocol to compute any arithmetic circuit $c \in C_M$ on k sets of inputs in parallel at a total cost of $O((M + 1)n^2 \log |F|)$ bits of communication.

Proof : Correctness of the protocol sketched above follows from the algebra. The communication bound follows since the initial multi-secret sharing requires $O(n^2 \log |F|)$ bits, and each multiplication subprotocol (both re-randomization and truncation) requires $O(n^2 \log |F|)$ bits as well. Privacy follows from Theorem 3.1, together with the claim that the multiplication subprotocol reveals no useful information to any coalition of up to $t - k + 1$ players. In that subprotocol, each player receives (points that interpolate to an otherwise random polynomial whose value at zero is the y coordinate of) a point on a polynomial of the form $\sum_{j=1}^k L_{e_j}(x)h(e_j) + q(x) \prod_{j=1}^k (x - e_j)$, where $q(x)$ is a random polynomial of degree $t - k$, and where $h(e_i)$ is the product of secret values for $1 \leq i \leq k$. From $t - k + 2$ of these points, the value of a linear combination of $h(e_1), \dots, h(e_k)$ could be determined. Any smaller set of points can only yield a linear combination of unknowns that includes one or more of the random coefficients of $q(x)$, since $q(x)$ has $t - k + 1$ coefficients. \square

3.3.4 Other Parallel Protocols

Many other secure computation protocols that rely on Shamir’s secret sharing scheme can be parallelized by modifying the algebra to accommodate multi-shares. In fact, all known non-cryptographic protocols tolerating a constant fraction of faulty players are parallelizable by our technique. In all cases, a multiplicative factor of k , for any $1 \leq k \leq t$, is gained in the amortized bit complexity, at the expense of an additive factor of k in the number of faulty players tolerated.

In particular, Ben-Or, Goldwasser, and Wigderson [22] present a t -resilient perfectly-secure protocol (BGW-res) for computing any arithmetic circuit in C_M over finite field F by $n \geq 3t + 1$ players with communication complexity $C_{BGW}(C_M) = O(Mn^2t^3 \log |F|)$ bits.

Theorem 3.3 (*Resilient Computation on Multi-Shares*) For $n \geq 3t + 1$, $t \geq k \geq 1$, there is a $(t - k + 1)$ -resilient protocol to compute any arithmetic circuit $c \in C_M$ (over a finite field) on k sets of inputs in parallel, at a communication complexity $C_{BGW}(C_M)$.

The BGW-res protocol adds verification and error-correction procedures to the protocol described in the preceding section. The verification procedures are always performed on single shares of single shares, which do not require modification when parallelized; these remain the same when performed on single shares of multi-shares. The error correction procedure is also independent of the number of secrets hidden by the polynomial; as long as the α_i ’s are chosen to be n th roots of unity, this procedure need not be changed for the parallel version.

Rabin and Ben-Or [128] (see also Beaver [9]) have a protocol to securely compute any arithmetic circuit t -resiliently, where $n \geq 2t + 1$, assuming broadcast, and allowing an exponentially small probability of error. With systematic changes to the truncation and randomization steps for

multiplication, k circuits can be $(t - k + 1)$ -resiliently computed at no increase in communication complexity, by substituting our multi-sharing scheme for Shamir secret sharing.

Other examples where systematic parallelization applies are the private and resilient protocols of Chaum, Crépeau, and Damgård [41], the constant-round protocols of Bar-Ilan and Beaver [6], and the general network topology protocols of Dolev, Dwork, Waarts, and Yung [61].

3.3.5 Computing “Similar Circuits” in Parallel

We also note that our parallel techniques are useful when circuits that are similar but not identical are to be computed on different sets of inputs. We have inexpensive protocols to split and join multi-shares. At roughly the cost of a single 2-ary multiplication subprotocol, multi-shared secrets can be converted into singly shared secrets, and singly shared secrets can be converted into multi-shared secrets, via secure multi-party protocols (either private or resilient). Thus the computations can be performed in parallel over those portions of the circuits that are identical, and performed separately over those portions that differ. If the circuits can be matched except for portions containing a small number of multiplication gates, then parallelization can still reduce communication complexity.

For example, suppose that the players want to join k singly-shared secrets $g_1(e'_1), \dots, g_k(e'_k)$, where each player i begins with the shares $g_1(\alpha_i), \dots, g_k(\alpha_i)$. The goal is to give each player i the multi-share $v_i = g(\alpha_i)$, where $g(e_j) = g(e'_j)$ for all j , $1 \leq j \leq k$. The algebra underlying the joining protocol is that

$$\vec{v} = \sum_{j=1}^k \vec{w} B_{e'_j}^{-1} \times \text{Chop}_{t-k+1} \times B_{e_j} \times M_{e_j}$$

where

$$w_i = \sum_{j=1}^k L_i(e'_j) g_j(\alpha_i).$$

Analogous methods can be used to recombine arbitrary collections of singly and multi-shared secrets into new single and multi-shares. The algebraic details are similar to the case described above, and are omitted. Splitting of multi-shares can then be viewed as a special case of recombining. It is also possible to “split” in practice simply by making copies of the multi-shares, with re-randomization if needed to “erase” unwanted secrets from each set of copies.

3.4 Direct Sum Problem for Communication Complexity of Secure Protocols

In the preceding section, we described a general technique for reducing the amortized bit complexity of a secure computation protocol, but at a small cost in number of faults tolerated. It is natural to consider whether this cost is necessary, i.e., can a reduction in amortized bit complexity ever be achieved without affecting any other parameters? This is a version of the “direct-sum problem.” The direct-sum problem for communication complexity has been considered recently by Feder, Kushilevitz, and Naor [68]. We present the first such results, one positive and one negative, for this problem for *secure computation*. We concentrate on the addition function.

Performing in parallel many private addition protocols over $GF(2)$ can decrease the required amount of communication at no cost to other parameters, as the following two lemmas indicate.

Lemma 3.2 (*Non-Parallel Private Lower Bound*) *Perfect t -private addition over $GF(2)$ among $n \geq 2t + 1$ players, where t is $O(n)$, requires $\Omega(n^2)$ bits of communication.*

Proof : It suffices to show that at least $nt/2$ messages must be exchanged. If fewer messages are exchanged, then some player u receives r messages and sends s messages, where $r + s \leq t$. Thus a group D of at most t players can learn all inputs and all outputs to the probabilistic TM that controls player u .

Assume that a random tape of length k is sufficient for each player in the protocol. The group D can check all possible 2^{k+1} settings of u 's input bit and random tape. For each setting, if the TM for player u outputs the exact sequence of s sent messages when it receives the exact sequence of r received messages, then that setting is said to be consistent.

If every consistent setting involves the same choice of input bit for u , then D can learn that secret input. Thus the adversary can learn the input bit of u with probability at least $\frac{(n-t)!t!}{n!} = C(n, t)^{-1}$, i.e., by correctly guessing which t players to corrupt (or with probability 1 if the protocol is oblivious). This violates the zero probability of disclosure needed for perfect privacy. If different consistent settings involve different input bits for u , then the protocol cannot guarantee the zero probability of output error needed for perfect privacy. \square

Lemma 3.3 (*Parallel Private Upper Bound*) *Perfect t -private addition (mod 2) among $n \geq 2t + 1$ players $n - t$ times in parallel is possible with $O(n^2 \log n)$ bits of communication.*

Proof : Each player multi-shares its $n - t$ secret input bits using the $(t, n; n - t, n)$ -multi-secret sharing scheme over $GF(2^m)$, where $2^m > 2n - t$ ("room" for n share locations and $n - t$ secret locations). Each player takes the xor of all received shares, and these results are interpolated to recover all $n - t$ xors. The proof follows from Theorem 3.1 and Lemma 3.1. \square

We summarize these two lemmas in the following theorem.

Theorem 3.4 (*Positive Direct Sum Result*) *Parallelization can reduce the amortized communication complexity of perfect t -private addition over $GF(2)$ among $n \geq 2t + 1$ players by a factor of $\Theta(\frac{n}{\log n})$.*

We note that the technique of Lemma 3.3 applies to any finite field, implying that t -private addition over $GF(q)$ can be performed $k \leq n - t$ times in parallel among $n \geq 2t + 1$ players with $O(n^2 \log(nq))$ bits of communication. We also note that essentially the same result as Lemma 3.2 was found independently by Kushilevitz [102], who gives exactly matching upper and lower bounds of $\frac{n(t+1)}{2}$ messages for multi-party private addition.

However, performing in parallel many unstopable addition protocols over any finite field, without reducing other parameters, cannot decrease the required amount of communication, as the following two lemmas indicate.

Lemma 3.4 (*Parallel Unstopable Upper Bound*) *Perfect oblivious t -unstopable addition among $n = 2t + 1$ players k times in parallel is possible with $O(kn^2 \log |F|)$ bits of communication.*

Proof : The protocol BGW-priv, when used to perform a single addition, is perfect and oblivious. It is actually t -unstopable, and requires $O(n^2 \log |F|)$ bits of communication. The proof of the lemma

then follows from the equivalence of performing k additions over a field of size m and performing a single addition over a field of size m^k (since $GF(m^k)$ can be constructed as $GF(m)[X]/f(X)$), where $f \in GF(m)[X]$ is any irreducible polynomial of degree k [135]). As with the proof of Lemma 3.3, $GF(m^k)$ must be large enough to have n share locations and one secret location, i.e., $|F|$ must be $\Omega(n^{1/k})$. \square

Lemma 3.5 (*Parallel Unstoppable Lower Bound*) *Perfect oblivious t -unstoppable addition among $n = 2t + 1$ players k times in parallel requires $\Omega(kn^2 \log |F|)$ bits of communication.*

Proof : As with the upper bound proof (by the computation over the extended field of size m^k and the additivity of the log function), it suffices to show an $\Omega(n^2 \log |F|)$ lower bound for a single addition. Assume that fewer than $n(n - 1) \log |F|/2$ bits are sent. Then some pair of players p and p^* send fewer than $\log |F|$ between themselves. Partition the rest of the players (excluding p and p^*), into subsets C and D , where $|C| = t - 1$ and $|D| = t$.

Let s_x denote the secret input of player x , and let s'_x denote the default input for player x if it drops out before the commit point. Let s_S denote the sum of secret inputs of subset S , i.e., $s_S = \sum_{x \in S} s_x$. Let $comm(X, Y)$ denote the transcript of all messages sent between players in X and players in Y .

Consider the following execution of the protocol. The players in C drop out immediately, sending no messages, and player p^* drops out after the commit point; all other players stay in until the end of the protocol. For this execution, D and p together can recover the appropriate output $s_p + s_{p^*} + s_D + s'_C$. Thus D and p together can recover s_{p^*} , since the other components of the output sum are known. This implies that, for some easy to compute function g , $s_{p^*} = g(s_D, s_p, comm(p^*, D), comm(p, p^*))$.

Since p and p^* exchange less than $\log |F|$ bits between themselves, there are less than $|F|$ possible values of $comm(p, p^*)$. Thus there are less than $|F|$ possible values of s_{p^*} for any given fixed $s_D, s_p, comm(p^*, D)$. This implies that D by itself can exclude some of the $|F|^2$ possible pairs of values for $[s_p, s_{p^*}]$ as being impossible. Since $|D| = t$, this violates the secrecy requirement for perfect unstopability of the protocol. \square

These two lemmas are summarized by the following theorem.

Theorem 3.5 (*Negative Direct Sum Result*) *Parallelization cannot reduce the communication complexity for perfect oblivious t -unstoppable addition (over a finite field) among $n = 2t + 1$ players.*

3.5 Communication complexity of detecting protocols

Protecting fully against malicious behavior can be quite expensive. For example, the lowest bit complexity for t -resilient computation, where t is a constant fraction of n , is due to Ben-Or, Goldwasser, and Wigderson [22]: $O(Mn^5 \log |F|)$ bits to t -resiliently compute a circuit in C_M (when $n \geq 3t + 1$). By contrast, the lowest bit complexity for t -private computation of a circuit in C_M , where t is a constant fraction of n ($n \geq 2t + 1$), is also due to Ben-Or *et al*: $O(Mn^2 \log |F|)$ bits. However, if malicious behavior is not frequent, then it may be sufficient to merely identify cheating when it occurs. Repeating the computation when cheating is detected, possibly switching to a resilient protocol, insures that all computations are eventually done correctly. This is advantageous if computation that identifies cheating can be more efficient than computation that corrects cheating.

In fact, t -detectability, where t is a constant fraction of n , and M is non-constant, can be achieved at a significant decrease in communication complexity. The protocol that satisfies the following theorem has a bit complexity that is within a log factor of the most efficient known protocol for general t -private computation (when t is a constant fraction of n), when M is sufficiently large.

Theorem 3.6 *For $n \geq 3t + 1$, $c \in C_M$, there is a $(t - 1)$ -detecting protocol to compute c with error probability ϵ , with $O((n^5 + Mn^2) \log(|F|(n + M)\epsilon^{-1}))$ bits of communication.*

Before giving the proof of this theorem, we develop some basic facts about polynomials over a finite field.

Fact 3.1 *If $t' > t$ points lie on a degree t polynomial, then they lie on no other polynomial of degree less than t' .*

Proof : If t' points lie on polynomials $p(x)$ and $q(x)$, then $p - q$ has at least t' zeros, and thus at least one of p and q must be of degree at least t' . \square

Let $p(x_1, \dots, x_n)$ be a polynomial over a finite field F of size m . Define the “degree” of p to be the largest exponent of any variable in p . Note that this is non-standard usage for the degree of a multinomial, ordinarily defined to be the largest sum of exponents of a term. Sometimes the expressions “maximum degree” and “total degree” are used in the literature to describe these two quantities.

Let $C[x_1, \dots, x_n]$ be an arithmetic circuit over F , $|F| = m$, composed of binary multiplication gates, binary addition gates, and unary scalar multiplication gates; C has one input wire associated to each x_i , and a single output wire. Each circuit $C[x_1, \dots, x_n]$ computes a unique function from $F^n \rightarrow F$ in the obvious way; let $f_C(x_1, \dots, x_n)$ denote this function. Each circuit $C[x_1, \dots, x_n]$ can be expressed as a formal polynomial over F in the obvious way; let $p_C(x_1, \dots, x_n)$ denote this polynomial. By the following fact, it suffices to consider circuits whose formal polynomials have “degree” less than m .

Fact 3.2 *Every function over F can be expressed as a polynomial of “degree” less than $|F|$.*

Proof : For every $a \in F$, $a^m = 1$. \square

Next we prove an upper bound on the number of zeros of a multivariate polynomial over a finite field that, while not the best possible, is sufficient for our needs. This result is also a consequence of a theorem (relating number of zeros to total degree) due to Zippel [142] and independently to Schwartz [131].

Lemma 3.6 *If $p(x_1, \dots, x_n)$ has “degree” d over F , $|F| = m$, then $p(x_1, \dots, x_n) = 0$ has at most ndm^{n-1} solutions.*

Proof : Let $z(n, d)$ be the number of solutions to $p(x_1, \dots, x_n) = 0$ over F , where p has n variables and is “degree” d . For any assignment to the first $n - 1$ of the variables, either $p(v_1, \dots, v_{n-1}, x_n) \equiv 0$ or $p(v_1, \dots, v_{n-1}, x_n)$ is a single variable polynomial of degree d over F . In the first case, x_n can take on any value in F to satisfy $p = 0$. In the second case, x_n can take on the value of any root of $p(v_1, \dots, v_{n-1}, x_n)$ to satisfy $p = 0$.

The first case occurs for at most $z(n-1, d)$ assignments of the first $n-1$ variables, as can be seen by rewriting the polynomial as $p(x_1, \dots, x_n) = p_0(x_1, \dots, x_{n-1}) + x_n p_1(x_1, \dots, x_{n-1}) + \dots + x_n^d p_d(x_1, \dots, x_{n-1})$, where each p_i is “degree” at most d with $n-1$ variables. For a given assignment, each $x_n^i p_i$ term must be zero (else $p(v_1, \dots, v_{n-1}, x_n) \neq 0$). At least one of the x_n^i terms must be present (else p would only have $n-1$ variables). The upper bound then follows.

The second case occurs for at most m^{n-1} assignments of the first $n-1$ variables (trivially). Since there are at most d distinct roots of any degree d single variable polynomial (i.e., since $z(1, d) \leq d$), we have the following:

$$z(n, d) \leq z(n-1, d)m + m^{n-1}d$$

Expanding, we have

$$\begin{aligned} z(n, d) &\leq (z(n-2, d)m + m^{n-2}d)m + m^{n-1}d = z(n-2, d)m^2 + 2m^{n-1}d \\ &\leq \dots \leq z(1, d)m^{n-1} + (n-1)m^{n-1}d \leq dm^{n-1} + (n-1)m^{n-1}d = dnm^{n-1}. \end{aligned}$$

This completes the proof. \square

Lemma 3.7 *If $C[x_1, \dots, x_n]$ is an arithmetic circuit over F , and if $k > 1$, then there is a finite field F' of size $|F|^k$ over which C computes the same function when considered as an arithmetic circuit over F .*

Proof : Take F' to be $F[x] \setminus q(x)$ for some irreducible single variable polynomial $q(x)$ of degree k over F . The natural mapping $\phi : F \rightarrow F'$ taking values in F to constant (degree zero) polynomials in F' is a monomorphism. \square

Definition 3.1 *If $C[x_1, \dots, x_n]$ is an arithmetic circuit over F to compute function $f_c(x_1, \dots, x_n)$, then let $f_c(x_1, \dots, x_n)[g \rightarrow g - y]$ denote the function computed by C if the output of gate g in C is decremented by y .*

Lemma 3.8 *For all arithmetic circuits $C[x_1, \dots, x_n]$ and all gates g in C , there exists a polynomial $p_{C,g}(x_1, \dots, x_n, x_{n+1})$ such that $p_{C,g}(x_1, \dots, x_n, y) = f_c(x_1, \dots, x_n)[g \rightarrow g - y]$, where the “degree” of $p_{C,g}$ is equal to the “degree” of p_C .*

Proof : A circuit C' to compute $f_c(x_1, \dots, x_n)[g \rightarrow g - y]$ can be derived from C (disconnect the output wire of g ; insert an input wire for y ; insert a scalar multiplication gate with input y and output $-y$; insert an addition gate with inputs $-y$ and the output of g ; connect the output of the addition gate to what used to be connected to the output of g). The formal polynomial $p_{C'}(x_1, \dots, x_n, y)$ has the same “degree” as p_C since the output of g has “degree” at least 1. \square

Now we can give the proof of the main Theorem.

Proof : (Theorem 3.6) Here is the error-detecting computation protocol. The secret inputs are s_1, \dots, s_n , the share locations are $\alpha_1, \dots, \alpha_n$, and the share locations are 0, 1. All computations are actually performed over an extension field F' of F , where $|F'| = |F|^\lambda$ (see Lemma 3.7). The choice of λ depends upon the desired probability ϵ of undetected cheating; it suffices to choose $\lambda \geq \lceil 1 + \frac{\log(Mn(n+1)\epsilon^{-1})}{\log|F|} \rceil$. (n is the number of players, M is the number of 2-ary multiplication gates in the arithmetic circuit).

Initialization Phase

1. Each player i finds an otherwise random degree $t - 1$ polynomial $p_i(x)$ such that $p_i(0) = s_i$, and gives to each player j the share $s_{ij} = p_i(\alpha_j)$ using a protocol for Verifiable Secret Sharing (VSS), e.g., as in [22].
2. Each player i finds n random degree $t - 1$ polynomials $r_{i1}(x), \dots, r_{in}(x)$, and gives to each player j the shares $r_{i1j} = r_{i1}(\alpha_j), \dots, r_{inj} = r_{in}(\alpha_j)$ using VSS. The l th “check input” will be $\sum_{k=1}^n r_{kl}(1)$.
3. Each player i finds v_{1i}, \dots, v_{ni} where $v_{li} = (1 - \alpha_i)s_{li} + \alpha_i(\sum_{j=1}^n r_{jli})$. These values are player i 's multi-shares of n polynomials of degree t that pass through the secret inputs at $e_1 = 0$ and the check inputs at $e_2 = 1$.

Computation Phase

At the beginning of the Computation Phase, each player holds a multi-share corresponding to each input of the arithmetic circuit.

1. To evaluate a 2-ary addition gate, each player privately adds its own multi-shares of the corresponding inputs to the gate. The result is considered to be the multi-share corresponding to the output of the addition gate.
2. To evaluate a scalar multiplication gate, each player privately finds the product of the scalar and its own multi-share of the corresponding input to the gate by the appropriate scalar. The result is considered to be the multi-share corresponding to the output of the scalar multiplication gate.
3. To evaluate a 2-ary multiplication gate, the players proceed as follows. Assume that the multi-shares of the corresponding inputs to the gate held by player i are $left_i$ and $right_i$.
 - (a) Each player i privately finds $temp_i = left_i \times right_i$.
 - (b) Each player i chooses a random polynomial $q_i(x)$ of degree $2t - 2$ and gives to each player j the value $\beta_{ij} = q_i(\alpha_j)$.
 - (c) Each player i privately finds $w_i = temp_i + \alpha_i(\alpha_i - 1)(\sum_{j=0}^{n-1} \beta_{ji})$.
 - (d) The players cooperate to compute e_3 and to give each player i the value v_i , where

$$(v_1 \cdots v_n) = \sum_{j=1}^2 [(w_1 \cdots w_n) \times B_{e_3}^{-1} Chop_{2t} B_{e_3} B_{e_j}^{-1} Chop_{t-1} B_{e_j} M_{e_j}]$$

using the following steps to accomplish this.

- i. Each player i chooses a random polynomial $p_i(x)$ of degree t such that $p_i(0) = w_i$.
- ii. Each player i gives to each player j the value $p_i(\alpha_j)$.
- iii. Each player i sends to each player j $s_{ij} = r_i(\alpha_j)$ for some random degree t polynomial $r_i(x)$.
- iv. Each player i sends $s'_i = \sum_{j=1}^n s_{ji}$ to all players.
- v. Each player privately finds e_3 to be the value at 0 of the degree t polynomial through s'_1, \dots, s'_n (complaining if degree is bad).

vi. Each player i privately finds, for $k = 1, 2$ the values

$$(p_1(\alpha_i) \cdots p_n(\alpha_i)) \times B_{e_3}^{-1} Chop_{2t} B_{e_3} B_{e_k}^{-1} Chop_{t-1} B_{e_k} M_{e_k} = (v_{1ki} \cdots v_{nki})$$

- vii. Each player i gives to each player j the value $v_{ji} = v_{j1i} + v_{j2i}$.
viii. Each player i privately finds v_i as the value at 0 of the degree t polynomial that interpolates $(\alpha_1, v_{i1}), \dots, (\alpha_n, v_{in})$ (complaining if degree is bad).

Revelation Phase

1. Each player i sends to each player j i 's multi-share out_i corresponding to the output wire of the circuit.
2. Through VSS and interpolation, the players recover all of the values $r_{ij}(1)$, $1 \leq i, j \leq n$.
3. Each player i complains unless all of the following are true:
 - (a) The final multi-shares out_1, \dots, out_n lie on a polynomial $p_{out}(x)$ of degree at most t ;
 - (b) All values shared using VSS were shared correctly; and
 - (c) The check output matches the result of the computation on the check inputs, i.e., $p_{out}(1) = f(\sum_{i=1}^n r_{i1}, \dots, \sum_{i=1}^n r_{in})$.
4. If there are no complaints, each player i takes the output of the circuit to be $p_{out}(0)$.

Cheating during the sharing or revelation phases of the VSS protocols will be caught always (using the VSS protocol of Ben-Or, Goldwasser, and Wigderson).

Sending incorrect values of out_i will be caught always, since the honest players' values uniquely determine a degree t polynomial.

Thus the only place for effective cheating is in the multiplication subprotocol. Cheating can occur in this subprotocol in steps 3b, 3dii, 3diii, 3div, and 3dvii. Assume that cheating occurs during a single multiplication subprotocol.

Cheating in step 3b can only affect the degree of the polynomial through the w_i values, but cannot affect the value of the polynomial at 0 and 1.

The steps 3diii-3dvi allow the players to agree on a random element. Cheating during this phase will either be detected (by affecting the degree of the polynomial through the s_i^t values), or will not bias the randomness of the selected element e_3 . Thus cheating in step 3div can not cause undetected error.

Cheating in step 3dvii will be caught in step 3dviii, since there is a unique degree t polynomial through the values received in step 3dvii by each honest player from the other honest players.

This leaves only cheating in step 3dii to consider. Cheating in this step enables the faulty processors to change their inputs to the truncation procedure (steps 3diii-3dviii). This can only cause all n points to be on a curve that is right at 1 and wrong at 0 if the new curve is a polynomial of degree at least $n - t + 1$. (by Fact 3.1), since the $n - t$ honest points lie on a degree $2t < n - t$ polynomial).

Assume that the vector of points that are used in steps 3diii-3dvii lie on the curve $a_{t'}x^{t'} + \dots + a_1x + a_0$, where $n - 1 \geq t' \geq n - t + 1 > 2t$. Multiplying the vector of points by $B_{e_3}^{-1} Chop_{2t} B_{e_3}$ is

equivalent to finding a new vector of points that lie on the curve $b_{2t}(x - e_3)^{2t} + \dots + b_1(x - e_3) + b_0$, where $a_t x^t + \dots + a_0 = b_t(x - e_3)^t + \dots + b_0$. The effect on the value of the curve at the check location 1 is to reduce it by $\xi(e_3) = (1 - e_3)^{2t+1}[b_t(1 - e_3)^{t-2t-1} + \dots + b_{2t+1}]$, where $n - 1 \geq \deg(\xi) > 2t$.

We can bound the probability that this reduction in the check value output at a single multiplication gate g will affect the check value output of the circuit. If $p_C(x_1, \dots, x_n)$ is the formal polynomial that computes the original circuit, the “degree” of p_C is less than $|F|$ (by Fact 3.2); thus there is a polynomial $p_{C,g}(x_1, \dots, x_n, y)$ that computes the same circuit when y is subtracted from the output of g , such that the “degree” of $p_{C,g}$ is equal to the “degree” of p_C (see Lemma 3.8). Thus the “degree” of $p_{C,g}(x_1, \dots, x_n, \xi(e_3))$ is at most $\deg(p_C)\deg(\xi) < |F|n$. Thus the “degree” of $\zeta(x_1, \dots, x_n, e_3) = p_{C,g}(x_1, \dots, x_n, \xi(e_3)) - p_C(x_1, \dots, x_n)$ is less than $|F|n$. Note that $\zeta(x_1, \dots, x_n, e_3)$ is zero if and only if the check value output of the circuit is unaffected by the tampering at gate g .

By Lemma 3.6, ζ has at most $(n + 1)(|F|n)|F'|^n$ zeros. Since the inputs to ζ (check values and e_3) are uniformly random in $|F'|$, the probability that ζ is zero is at most $\frac{|F|n(n+1)|F'|^n}{|F'|^{n+1}} = \frac{|F|n(n+1)}{|F'|}$.

When cheating occurs in $\kappa > 1$ multiplication subprotocols, the above analysis can be extended as follows. Let $G = g_1, \dots, g_\kappa$ be the gates at which cheating occurs. Let $e_{31}, \dots, e_{3\kappa}$ be the jointly created random values in steps 3diii-3dv. Let $p_{C,g}(x_1, \dots, x_n, y_1, \dots, y_\kappa)$ be the polynomial that computes what the circuit C computes when the output of each gate g_i is reduced by y_i . Note that $\deg(p_{C,g}) = \deg(p_C) < |F|$. Let $\xi + i(e_{3i})$ be the reduction in the output of gate g_i caused by cheating, $n - 1 \geq \deg(\xi_i) > 2t$ for all i . Thus the “degree” of $p_{C,g}(x_1, \dots, x_n, \xi_1(e_{31}), \dots, \xi_\kappa(e_{3\kappa}))$ is at most $\deg(p_{C,g})(\max_i \deg(\xi_i)) < n|F|$. Thus the polynomial $\zeta(x_1, \dots, x_n, e_{31}, \dots, e_{3\kappa}) = p_{C,g}(x_1, \dots, x_n, \xi_1(e_{31}), \dots, \xi_\kappa(e_{3\kappa})) - p_C(x_1, \dots, x_n)$ also has “degree” less than $n|F|$. By Lemma 3.6, ζ has at most $(n + \kappa)(n|F|)|F'|^{n+\kappa-1}$ zeros. Thus the probability that ζ is zero for uniformly random inputs is at most $\frac{(n+\kappa)n|F|}{|F'|}$; this is also the probability that the check output is unchanged.

The overall probability for undetected cheating is thus

$$\max_{\kappa_1 + \kappa_2 + \dots \leq M} \sum_i \frac{(n + \kappa_i)n|F|}{|F'|} = \frac{M(n + 1)n|F|}{|F'|}$$

which is at most ϵ when $|F'| = |F|^\lambda$ for $\lambda = \lceil 1 + \frac{\log(Mn(n+1)\epsilon^{-1})}{\log|F|} \rceil$.

□

Note that the adversary has a lot of control over the polynomial $\zeta(x_1, \dots, x_n, \xi(e_3))$, and can know exactly what it will be as a result of tampering in step 3dii. The adversary can change the faulty players’ inputs to the truncation procedure (steps 3diii-3dviii) as follows. Choose any degree t' polynomial $q(x)$ that is zero at all of the honest players’ share locations and zero at the check location, $n - 1 \geq t' \geq n - t + 1 > 2t$. Change each faulty player’s input to the truncation procedure from w_i to $w_i + q(\alpha_i)$, where α_i is faulty player i ’s share location. The polynomial $\xi(e_3)$ is completely determined by $q(x)$, and thus so is ζ .

For a worst-case scenario, suppose that the circuit computes the function $f(x_1, \dots, x_n) = (\sum_{i=1}^n x_i)x_j^{M-1}$. Suppose that the circuit begins with the computation of $\sum_{i=1}^n x_i$, and then performs M 2-ary multiplications sequentially. Further assume that player j is under the control of the adversary, so the value of x_j will be known in advance by the adversary. The adversary can

attempt to cheat during the first multiplication subprotocol, which finds the product of $\sum_{i=1}^n x_i$ and x_j . The adversary will know if the cheating was successful, since it can check whether the e_3 that was constructed during the subprotocol is a zero of the polynomial ξ that the adversary chose. If the cheating was successful, then the adversary is honest for the remainder of the computation. If unsuccessful, then the adversary cheats on the next multiplication subprotocol. For this one, however, the adversary chooses $q(x)$ slightly differently: $q(\alpha_i) = 0$ for each honest player's share location α_i , and $q(1) = -\xi(e_3)s_j$. If the adversary cheats successfully this time then check value will be restored to its correct value, and the adversary can be honest thereafter. The adversary can continue in this way, for up to M separate chances to cheat successfully.

3.6 Conclusion

We have initiated the study of the communication complexity of multi-party non-cryptographic secure computation protocols. Parallelization can often reduce the amortized bit complexity of existing protocols at a small price in fault tolerance. Allowing no reduction in fault tolerance, matching lower and upper bounds show that amortized bit complexity can be improved in one setting (privacy) but not in another (unstoppability). When malicious behavior is infrequent, detectability can offer as much protection as resilience, at significantly reduced bit complexity.

Many open questions remain in the general area of communication complexity for secure computation. Non-trivial lower bounds for resilient computation and for cryptographic protocols (e.g., allowing one-way functions or Oblivious Transfer) would be interesting. Tradeoffs among complexity measures also warrant further study. We have shown a tradeoff between bit complexity (multiplicative) and fault tolerance (additive); it is possible that other measures can be related as well (e.g., round complexity, level of security, error probability). Lastly, the resilient protocol of Beaver, Feigenbaum, Kilian, and Rogaway [14] has a bit complexity that is *independent* of circuit size, but is low in fault tolerance (i.e, can only tolerate $O(\log n)$ faulty players); whether fault tolerance can be improved while maintaining this independence remains open.

Chapter 4

Eavesdropping Games: A Graph-Theoretic Approach to Privacy in Distributed Systems

We initiate a graph-theoretic approach to study the (information-theoretic) maintenance of privacy in distributed environments in the presence of a bounded number of mobile eavesdroppers (“bugs”). For two fundamental privacy problems – secure message transmission and distributed database maintenance – we assume an adversary is “playing eavesdropping games,” coordinating the movement of the bugs among the sites to learn the current memory contents. We consider various mobility settings (adversaries), motivated by the capabilities (strength) of the bugging technologies (e.g., how fast can a bug be reassigned). We combinatorially characterize and compare privacy maintenance problems, determine their feasibility (under numerous bug models), suggest protocols for the feasible cases, and analyze their computational complexity.

4.1 Introduction

Motivation: Security of computation has triggered a number of fundamental studies in recent years, including foundations of cryptographic primitives, secure computation protocols, zero-knowledge proofs, software protection, and analysis of traffic patterns. Privacy of individuals in the presence of eavesdropping (Big Brother) is a fundamental issue in security of computation that has been dealt with in many ways. In this work we suggest a combinatorial approach to the problem of maintaining privacy in a network that is threatened by *mobile eavesdroppers*, i.e., by an adversary that can move its bugging equipment within the system. Mobile adversaries in the context of secure computation were introduced by Ostrovsky and Yung [122].

Unlike previous studies, this work takes a graph-theoretic perspective, to understand the relationship of network topology to the feasibility of fundamental privacy problems. We would like to characterize these fundamental problems combinatorially, and to analyze their complexity as well. Furthermore, we want to compare the difference in strength among the various eavesdropping mechanisms.

Model: The network is assumed to be a collection of simple nodes (processors/ switches/ link-

adaptors) that can store data, send and receive messages along the network links, and perform limited computation. The network's nodes act in rounds (synchronously), and are assumed to be vulnerable to eavesdropping. We remark that the assumption of node vulnerability can also model eavesdropping at edges in an actual network, but because of fiber-optics technology, future eavesdropping will likely take place not along links but at switches, bridges, routers and other actual network nodes.

We model the eavesdropping threat (adversary) as a dynamic phenomenon. There is a maximum number of nodes that can be under surveillance at any moment, but the actual nodes being targeted can change over time (e.g., each round). Once at a node, the (non-disrupting) bug gets to learn all incoming and outgoing messages and the memory contents. One can imagine that eavesdropping is possible at sites where intruders are physically present, manually tapping into the targeted locations. Alternatively, the eavesdropping threat might be centralized and remote, perhaps reconfigurable points of focus chosen arbitrarily from a spy satellite. Another possibility is that the eavesdroppers flow with the network message traffic (to neighbors only). This implies that we have to consider various *mobility settings* that define how to constrain bug movement between rounds.

Problems: When studying the privacy of information in a network, there are two security sub-fields to consider: communication security, i.e., protecting messages transmitted through the network, and data security, i.e., protecting information stored in the network. We consider problems from both sub-fields, i.e., *secure message transmission*, and *distributed database maintenance*. In the first problem a sender uses the network to send a private message to a receiver (with two variants according to whether sender and receiver are part of the network). In the second problem, data distributed among the network's nodes is to be maintained privately via communication of messages and updates. In both problems we assume that after each round of communication the adversary may move its bugs subject to the mobility setting.

The basic problems and the various capabilities of bugs (mobility settings) give us a rich spectrum of (parameterized) models that we investigate and compare.

Results: We show the following results:

- Secure Message Transmission
 - Complete characterization of feasibility for two variants, independent of mobility setting, depending only on certain *network connectivity properties*.
 - Computational complexity of determining feasibility for two variants (co-NP-complete for one, polynomial time for the other).
- Distributed Database Maintenance
 - Complete characterization of feasibility, sensitive to both mobility setting and network topology, in terms of *node search number*.
 - Computational complexity of determining feasibility in the general case (PSPACE-complete), and for several natural mobility settings (NP-complete for some, co-NP-hard for another).
 - Separation of number of bugs tolerated in different mobility settings (large additive and multiplicative gaps).

- Separation of adversary’s time to compromise privacy in different mobility settings (exponential versus polynomial rounds).

Related Work: Information-theoretic security for distributed tasks has been studied for many adversarial models involving non-mobile faults, beginning with Ben-Or *et al.* [22] and Chaum *et al.* [41]. A characterization of which boolean functions are privately computable on a complete network versus non-mobile eavesdroppers was given by Chor and Kushilevitz [55]. Variants of the problems we consider have been defined elsewhere. Secure message transmission was originally defined by Dolev *et al.* [61] under a mobile fault model, but assuming that many channels connect the sender and receiver. Database maintenance was defined by Ostrovsky and Yung [122] under a mobile fault model, but assuming broadcast and a complete network of untappable channels. Our work on distributed database maintenance shows reductions between privacy and generalizations of graph searching games, a topic that has received much attention (e.g., [36] [110] [100] [104] [25]).

Organization: In Section 4.2, we give the basic model of eavesdropping games. We consider the problem of secure message transmission in Section 4.3, and distributed database maintenance in Section 4.4. The sensitivity of the database problem to the choice of mobility setting is addressed in Section 4.5. Some open problems are given in Section 4.6.

4.2 Model of Eavesdropping Games

We consider the following synchronous game between a network and an adversary. Each node of the network is a processor that can do local computation, flip coins, store and erase information, and send messages to adjacent nodes. Some number of traveling eavesdropping “bugs” occupy nodes of the network, observing messages and memory contents without disrupting node behavior. The protocol that the network will execute is known to both the network and the adversary.

The adversary makes the first move in the game by choosing initial locations for bugs. The network makes a move in the game by executing a round of its protocol, and the adversary responds by reassigning the bugs (subject to some constraints). The network wins the game – and we say that the protocol is *private* – if the execution of the protocol leaks no useful information to the adversary.

Network Behavior: To execute a round of its protocol, each node in the network does the following:

1. Receive messages from neighboring nodes (except first round).
2. Flip coins (using an *on-line* source of randomness).
3. Compute messages to be sent.
4. Update local memory (storage and erasure).
5. Send messages to neighboring nodes (except last round).

We note that it is important that processors be able to erase information, and that their source of randomness be on-line. Otherwise, the adversary would know all past and/or future behavior of the network as soon as each node had been visited by at least one bug.

Adversary Behavior: To respond to a network round, the adversary does the following (after Step 5 of the round, and before Step 1 of the next round):

1. Coordinate the movement of the bugs subject to some set of constraints (“mobility setting”), possibly removing some bugs from the network, possibly placing some bugs onto the network, and possibly moving some bugs directly from one node to another.
2. Begin eavesdropping again at the start of the network’s next round.

Notice that if a bug is removed from a node at the end of a round, it sees the messages sent by its old location but not to its old location during that round; and if a bug is placed at a node at the end of a round, it sees the messages sent to its new location but not by its new location during that round.

Mobility Setting: A mobility setting is a collection of constraints on the allowable movement of bugs in the network. Different mobility settings yield different games between the network and the adversary, and thus different privacy problems.

We model all mobility settings to be oblivious (i.e., constraints on bug movement cannot depend on observed behavior of the network itself) and efficient (i.e., a judge with polynomially bounded resources in the size of the network can monitor the movement of bugs and decide legality for each round). We note, however, that a mobility setting actually need not be either oblivious or efficient. If a protocol leaks some information to an adversary that bases its actions on what is seen at the nodes, then it also leaks some (possibly much smaller) positive amount of information to an oblivious adversary, which would violate information-theoretic security. If the rules governing bug movement cannot be judged from round to round in polynomial time, then all of our results still hold except for one (Claim 4.3 in Section 4.4).

The weakest possible mobility setting allows no bug movement whatsoever during the protocol; this is called the “static” setting. The strongest possible setting allows any movement, i.e., every bug can move to an arbitrary location after step 5 of every round of the protocol; this is called the “frequent parallel hopping” (F/P/H) setting.

We (intuitively) say that a mobility setting is “natural” if the constraints on the future movement of a bug do not depend on the past or current behavior of other bugs, or in a “complex” way on the past behavior of that bug. Natural settings can be defined, e.g., by limiting the frequency with which bugs can move (e.g., each bug can move only every k rounds) or by limiting their range of motion (e.g., each new location must be within k edges of its old location), or by limiting the number of bugs that can move each round, or by stipulating a delay between the removal of a bug and its placement at a new location, or by restricting each bug to a subset of the network. We will identify and discuss some specific natural settings in Section 4.5.

4.3 Secure Message Transmission

4.3.1 Problem

In the secure message transmission problem (isolated as a fundamental primitive for secure computation by Dolev *et al.* [61]), two parties wish to communicate by using the graph (network of processors with eavesdropping bugs) as a communication channel. Suppose that party x has a

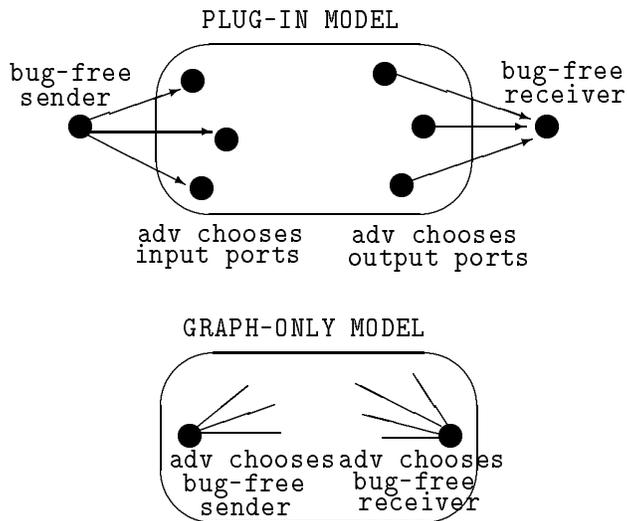


Figure 4.1: Plug-In and Graph-Only Models for Secure Message Transmission

secret message (element of a finite group) to send to party y . Party x can send messages to only p nodes of the graph (along “input ports”), and party y can receive messages from only p' nodes of the graph (along “output ports”). We call this the “plug-in” model for secure message transmission because x and y are “plugged into” the graph on which the mobile bugs are active (and to distinguish it from the upcoming “graph-only” model).

The adversary in this case has control over the movement of bugs on the graph, and has control over the location of the ports. The t bugs are somewhere on the graph at the start of the protocol, and can only move on the graph, i.e., cannot occupy x or y . We ask whether x, y and the nodes of the graph can behave so as to allow y to learn the message without the adversary learning anything about the message (see Figure 4.1).

Specifically, the nodes and bugs behave as in the basic model of eavesdropping bugs, with the following exceptions. In the first step of every round, *including the first round*, nodes that are input ports may receive messages from the sender x . In the fifth step of every round, *including the last round*, nodes that are output ports may send messages to the receiver y .

Secure message transmission is a primitive from which more general protocols can be built [61]. For example, if a number of parties are all plugged into a graph, then they can use the graph as a complete network of untappable channels. In this way, they can compute any function over a finite field privately using protocols secure against a passive adversary [22] [41].

To describe our results about message transmission in the plug-in model, we need the following definition. A graph $G = (V, E)$ is “ (L, U, d) -connected” if every subset of nodes of size between L and U has at least d neighbors, i.e.,

$$S \subset V, L \leq |S| \leq U \implies |ng(S)| \geq d.$$

4.3.2 Characterization of Feasibility

Success depends only on a connectivity requirement of the graph, and not on the choice of setting.

Theorem 4.1 *Secure message transmission in the (p, p') plug-in model is possible in setting σ versus t bugs on a graph G iff $p, p' > t$ and G is $(p - t, n - p', t + 1)$ -connected.*

Proof : The following two-phase protocol suffices versus an adversary in the (strongest possible) F/P/H setting. Initially, the sender x inputs a random share m_i of his message m on each input port i , such that $\sum_{i=1}^p m_i = m$. During each round of the first phase (except the last), each node i chooses a random value to send to each neighbor, treating the receiver as a neighbor if i is an output port; each node then updates its stored value (initially zero) by adding the values received at the start of the round (if any) and subtracting the values that will be sent at the end of the round (if any). In the second phase, which begins with step 5 of the last round of the first phase, the nodes simply propagate their final shares towards (and out) the output ports. The receiver finds the message to be the sum of all unique messages received from the output ports.

It can be shown that the adversary cannot learn any useful information about the message from the values that it sees in the first phase together with all the available final shares in the second phase, i.e., some necessary additive share is always lost through an output port. For example, if the first phase lasts $n - p - p' + t + 2$ rounds, then the balanced connectivity condition implies the existence of a sequence of nodes x_1, \dots, x_k such that x_1 is an input port, x_k is an output port, and each x_i is unoccupied during Round i , $1 \leq i \leq k \leq n - p - p' + t + 1$. In this case, the adversary learns no information about the initial share m_1 from the sender to x_1 , the value sent from x_i to x_{i+1} at the end of Round i for each $1 \leq i \leq k - 1$, and the value sent from x_k to the receiver at the end of round k . Without this information, each possible message m is equally likely from the adversary's point of view.

For the reverse direction, we show that the adversary can always succeed in the (weakest possible) static setting if the conditions are not met. If $p \leq t$ ($p' \leq t$), the adversary succeeds by occupying all input (output) ports throughout the protocol. If G is not $(p - t, n - p', t + 1)$ -connected, then there exists a set $S \subset V$ such that $p - t \leq |S| \leq n - p'$ while $|ng(S)| \leq t$. The adversary chooses input port locations $I = I_1 \cup I_2 \cup I_3$ such that $I_1 \subseteq S$, $|I_1| = p - t$, $I_2 = ng(S)$, $I_3 \subseteq V - S - ng(S)$, $|I_3| = t - |ng(S)|$ (possibly empty). The adversary chooses output port locations $O \subseteq V - S$. The adversary can then succeed by occupying $I_2 \cup I_3$ throughout the protocol. \square

Claim 4.1 : *The decision problem for secure message transmission feasibility in the plug-in model is co-NP-complete*

Proof : The proof is by reduction from the complement of MIN EDGE-CUT INTO EQUAL-SIZED SUBSETS [79]. Let (G, k) be an instance of this latter decision problem, i.e., does there exist a set of fewer than k edges whose removal splits $G = (V, E)$ into two subgraphs with $|V|/2$ nodes each? We construct an instance of the former decision problem as follows.

Let $n = |V|$, and let Δ be the maximum degree of any vertex in G . Choose $\lambda > \max(k, \frac{n\Delta}{2})$. Let $G' = (V', E')$, where (1) for each vertex $v \in V$, there are vertices $v_1, \dots, v_\lambda \in V'$; (2) for each edge $e \in E$, there is a vertex $v_e \in V'$; (3) for each vertex $v \in G$ with adjacent edges e_1, \dots, e_d , there are edges in E' connecting every v_1, \dots, v_λ to every $v_1, \dots, v_\lambda, v_{e_1}, \dots, v_{e_d}$. The subgraph

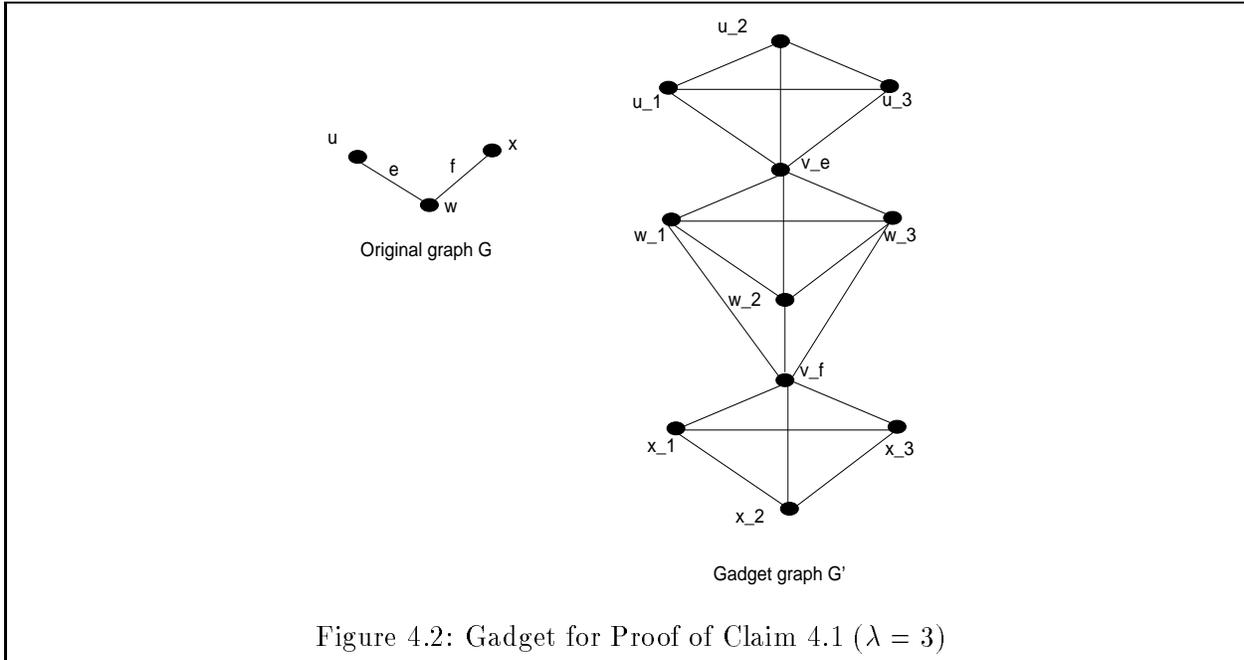


Figure 4.2: Gadget for Proof of Claim 4.1 ($\lambda = 3$)

in G' associated with a vertex v and its adjacent edges in G is called the “cluster” for v in G' . See Figure 4.2. We claim that G has an equal-sized edge cut of size less than k iff G' is not $(\frac{n\lambda}{2}, \frac{n(\lambda+\Delta)}{2}, k)$ -connected.

Suppose G has an equal-sized edge cut of size less than k . The set of nodes in G' corresponding to these edges is a node cut in G' . Each side of the cut in G' has exactly $\frac{n}{2}$ clusters. Since any cluster must have at least λ and at most $\lambda + \Delta$ nodes, each side of the cut must have between $\frac{n\lambda}{2}$ and $\frac{n(\lambda+\Delta)}{2}$ nodes, i.e., this node cut is balanced.

Suppose G' has a balanced node cut of size less than k . Then each side of the cut has between $\frac{n\lambda}{2}$ and $\frac{n(\lambda+\Delta)}{2}$ nodes. Each side of the cut must have exactly $\frac{n}{2}$ clusters, since the largest size of $\frac{n}{2} - 1$ clusters is less than $(\frac{n}{2} - 1)(\lambda + \Delta) < \frac{n\lambda}{2}$. Thus the corresponding edge cut in G , removing every edge e such that v_e was in the cutset for G' (and ignoring any other nodes in the cutset for G'), is an equal sized edge cut for G . \square

When $\max(p, p') < \lceil \frac{n+t}{2} \rceil$, we can restate the balanced connectivity requirement in terms of separators: We require that G not have an $(n - \min(p, p'))$ separator of size t .

When the port locations are fixed beforehand (or when p and p' are constant) the decision problem becomes polynomial time. For example, when the port locations are fixed then deciding feasibility reduces to deciding an instance of s - t connectivity.

Claim 4.2 *Given fixed port locations, consider the new graph G^* whose nodes are the nodes of G plus the sender and receiver, and whose edges are the edges of G plus the edges connecting sender and receiver to ports. Secure message transmission from sender to receiver through G is possible versus t bugs if and only if there are more than t node disjoint paths in G^* connecting sender and receiver.*

Proof : One direction of the proof is trivial, since private transmission is impossible when all possible paths are occupied throughout. The other direction can be easily shown for a transmission protocol where the sender overwhelms the bugs while the nodes of the network are simple relays. Let L be the sum of the lengths of $t + 1$ node disjoint paths from sender to receiver in G^* . The sender splits his message into $(t + 1)(L + 1)$ additive shares, and sends a different share to each input port each round (until the shares run out). The nodes simply propagate shares along the disjoint paths, with no splitting or recombining, and with no activity at all by nodes off the paths. The bugs get overwhelmed, since $t + 1$ new shares enter the graph every round while at most t shares are seen by bugs every round. The number of unseen shares thus increases by at least one every round. After $L + 1$ rounds, there have been at least $L + 1$ shares that have entered the network but have not been seen yet by any bug. Since the paths can only hold L shares, some share must have passed completely through and out the network without ever being seen. \square

The idea of this proof can also be used for Theorem 4.1. The processor nodes of the network become simple store-and-forward relays, while the work of the sender increases from one round of transmission to many ($O(n)$) rounds.

Graph-Only Model: We can consider the “graph-only” model of secure message transmission, in which the sender and receiver are two of the nodes in the graph. Here we stipulate that the adversary cannot eavesdrop on the sender or receiver even when bugs occupy those nodes (although, in fact, we don’t care whether the sender is shielded after the first round). Note that this problem is uninteresting without assuming some shielding of both sender and receiver, since eavesdropping at either site makes privacy impossible. The adversary should learn nothing about the message being sent for every possible choice of sender and receiver.

Theorem 4.2 *Secure message transmission in the graph-only model is possible in setting σ versus t bugs on a graph G iff G is $(t + 1)$ -connected.*

Proof : If G is not $(t + 1)$ -connected, then there exist nodes x, y such that every path from x to y passes through a cutset C , $|C| \leq t$. Secure message transmission from x to y is impossible in the static setting if the adversary occupies C throughout the protocol.

If G is $(t + 1)$ -connected, and x wishes to send a message to y , the protocol and proof from Theorem 4.1 can be modified in a straightforward manner. The first phase of a two-phase protocol ends when y has received a critical additive share of the message about which the adversary has no information. \square

Claim 4.3 ([63]) *The decision problem for secure message transmission feasibility in the graph-only model is in P .*

Example: Let G be a finite rectangular mesh, $t = 2$ bugs, $p = p' = 4$ input and output ports. Secure message transmission is possible for this example in the plug-in model, but not in the graph-only model.

4.4 Distributed Database Maintenance

4.4.1 Problem

In the distributed database maintenance problem (first considered as a secure computation task by Ostrovsky and Yung [122]), one or more secrets (elements of a finite group) are initially shared among the nodes of the graph. Then t bugs are introduced into the graph, and some protocol is executed ongoingly. At the end of every round, the secrets must be uniquely recoverable from the information held at the nodes of the graph. At the end of every round, the adversary must not be able to predict the secrets with a success probability better than random guessing.

4.4.2 Characterization of Feasibility

Node searching is a graph game with a long history [36] [123] [110] [100]. We generalize earlier formulations of the game as follows. At the start of the game, all edges of a graph are contaminated and guards occupy some nodes; the contaminated edges may be viewed as the possible locations of a fugitive that the guards are attempting to capture. At the start of each round (except the first round), one or more guards may move. Next, every contaminated edge that is adjacent to two occupied nodes becomes cleared. Finally, every clear edge that is adjacent to a contaminated edge via an unoccupied node becomes contaminated. The guards win the game if they succeed in clearing all edges of the graph (guaranteeing the capture of the fugitive).

Interesting variants of node searching may depend on the allowable movement of the guards. As with our bug settings, any restriction on the allowable movement of guards produces a new variant of the node searching problem. In fact, each setting for constraining the movement of bugs can also be viewed as a setting for constraining the movement of an equal number of guards: A guard can move at the beginning of a round if the corresponding bug could move before the start of that round.

This definition of node searching is more general than earlier formulations (e.g., [100]). In earlier work on node searching, the contamination is typically assumed to spread as far as possible after each round of guard movement (i.e., stopping only where blocked by guards); this may be viewed as searching for a very fast fugitive. We can model this behavior by choosing a setting in which guard movement is only allowed every $|V|$ rounds. This enables the contamination to spread as far as it possibly can before guards may move again.

If the guards are restricted to move as in setting σ , then we let $ns_\sigma(G)$ denote the minimum number of guards required to fully clear the graph G . Note that σ can be the F/P/H setting, the static setting, or any other collection of constraints on the allowable movement of bugs in the network.

The following theorem gives the relation between this generalized node search game and the problem of maintaining a distributed database.

Theorem 4.3 *Distributed database maintenance is possible in setting σ versus t bugs in graph G iff $t < ns_\sigma(G)$.*

Proof: First we show that if $t < ns_\sigma(G)$, then the database can be maintained indefinitely. Wlog, assume the database consists of a single secret from a finite group \mathcal{G} . The nodes can succeed by performing the following protocol. At the start, each node x_i holds an additive share s_i of the secret $s = \sum_i s_i$, for some $s \in \mathcal{G}$. Each round, each node x_i does the following:

1. Receive message r_{ji} from each neighbor x_j (except first round).
2. Flip $|ng(x_i)||\mathcal{G}|$ coins.
3. Use coin flips to determine uniformly random group elements r'_{ij} for every neighbor x_j . These are the messages to be sent at the end of the round.
4. Update local memory as follows:
 - (a) Compute $s_i^* = s_i + \sum_j r_{ji}$.
 - (b) Compute $s'_i = s_i^* - \sum_j r'_{ij}$.
 - (c) Erase everything except new share s'_i and messages to be sent, unless it is time to reconstruct s (in which case, s_i^* is saved).
5. Send message r'_{ij} to each neighbor x_j .

Suppose that the adversary could learn the secret after some number of rounds k . Consider the movement of guards in the generalized node search game identical to the movement of the bugs in the eavesdropping game for these k rounds. Since $t < ns_\sigma(G)$, there must be a contaminated edge at the end of these guard movements. In fact, by the definition of node searching, there must be a sequence of edges e_0, e_1, \dots, e_k such that (a) for every $0 \leq j \leq k$, either $e_j = e_{j+1}$ and one endpoint of e_j is unoccupied during round $j+1$, or e_j adjacent to e_{j+1} and their common endpoint is unoccupied during round $j+1$; and (b) for every $1 \leq j \leq k$, e_j is contaminated by the end of round j (i.e., contaminated at the start of round $j+1$).

Let x_0, x_1, \dots, x_k be a sequence of nodes defined as follows. If $e_j = e_{j+1}$ then x_j is any endpoint of e_j that is unoccupied during round $j+1$, and if e_j adjacent to e_{j+1} then x_j is their common endpoint. It is easy to verify that (a) for every $0 \leq j \leq k$, either $x_j = x_{j+1}$ or x_j is adjacent to x_{j+1} ; and (b) for every $0 \leq j \leq k$, x_j is unoccupied during steps 1-5 of round $j+1$.

The adversary does not see the activities (steps 1-5) at node x_j during round $j+1$, for all $0 \leq j \leq k-1$. Suppose that the adversary could somehow see all other activity that occurred (e.g., could occupy every node except x_j during round $j+1$, for all $0 \leq j \leq k-1$). Then the adversary doesn't see the initial share held at x_0 , or the messages sent along all true edges (x_j, x_{j+1}) , $x_j \neq x_{j+1}$, but sees all other relevant information (excluding intermediate erased computations at each x_j during round $j+1$). There are $|\mathcal{G}|^{\lambda+1}$ possibilities for the unseen information, where λ is the number of true edges (i.e., the number of j 's such that $x_j \neq x_{j+1}$). Each of the $|\mathcal{G}|$ possible secret being maintained by the database is consistent with exactly $|\mathcal{G}|^\lambda$ of these possibilities (i.e., every secret is consistent with all possible message sequences and one possible initial share held at x_0). Thus the adversary has no information about the secret maintained by the database at the end of k rounds.

For the reverse direction, it suffices to show that the adversary can succeed whenever $t \geq ns_\sigma(G)$. The following facts about eavesdropping games are useful:

1. The value held by a node at the end of round i is completely determined by the value held by the node at the end of round $i-1$, the messages sent to the node at the end of round $i-1$ (if any), and the coin flips of the node in round i .

2. The messages sent by a node at the end of round i are completely determined by the value held by the node at the end of round $i - 1$, the messages sent to the node at the end of round $i - 1$ (if any), and the coin flips of the node in round i .
3. If a node is occupied during round i , then the adversary knows the value held by the node at the end of round i , and the messages sent by the node at the end of round i .
4. The protocol must be correct for every possible collection of coin flips by the nodes.

When $t \geq ns_\sigma(G)$, there is a legal movement of guards in the node search game that clears every edge of G . We claim that the corresponding movement of bugs in the eavesdropping game allows the adversary to learn the secret being maintained by the database.

In the node search game, call a node clear if it is not adjacent to any contaminated edge. In the eavesdropping game, call a node clear if the corresponding movement of guards for bugs would clear the node in the node search game. Using facts 1-3, we can show the following round invariant: If the adversary can guess the coin flips through round m of the eavesdropping game, then the adversary knows the value held at, the messages sent from, and the messages sent to every clear node at the end of round m . This is trivially true at the start of the protocol, when no nodes are clear. Assume it is true through the end of round m .

Let x be a node that is clear at the end of round $m + 1$. If x was clear at the end of round m , then the adversary knows the value held at x and the messages sent to x at the end of round m (by the induction hypothesis). By facts 1 and 2, then, the adversary knows the value held at x and the messages sent by x at the end of round $m + 1$. Since x remains clear, every neighbor of x must either be clear at the end of round m or occupied by a bug during round $m + 1$. In either case, the adversary knows the message sent to x by each neighbor of x at the end of round $m + 1$ (either by the induction hypothesis together with fact 2, or by fact 3). Thus the adversary knows all messages sent to x at the end of round $m + 1$.

Otherwise, x is clear at the end of round $m + 1$ and adjacent to a contaminated edge at the end of round m . To be cleared, x must be occupied during round $m + 1$, and thus the adversary knows the value held at x and the messages sent by x at the end of round $m + 1$ (by fact 3). Furthermore, each neighbor of x must be either occupied during round $m + 1$ or clear at the end of round m . In either case, the adversary knows the message sent to x by the neighbor at the end of round $m + 1$ (either by fact 3, or by the induction hypothesis together with fact 2).

Given this round invariant, the adversary knows the value held by every node once the graph is fully cleared, if all coin flips can be guessed. By fact 4, any set of coin flips that is consistent with what the adversary has seen can be used (i.e., any set of consistent coin flips *could* have actually occurred, and the protocol must be correct for that set of coins), and thus the adversary learns information from which the real secret could be recovered. \square

The following claim shows that the decision problem for distributed database maintenance versus an *arbitrary* mobility setting is PSPACE-complete. As noted in Section 4.2, this is the only result that relies on the efficient “judgability” of mobility settings (although it suffices for judging to be in PSPACE).

Claim 4.4 *The decision problem for distributed database maintenance is PSPACE-complete.*

Proof : The decision problem is clearly in non-deterministic PSPACE, since a successful search can be guessed one step at a time; thus it is in deterministic PSPACE (by Savitch’s Theorem). [Note that this direction makes use of the efficiency requirement for mobility settings, although it remains true if the judge is PSPACE instead of P-time.]

It suffices to exhibit a graph, mobility setting, and number of guards for which it is PSPACE-hard to decide if the guards can successfully clear the graph. We do this by a reduction from QBF.

Let $(Q_1x_1) \cdots (Q_nx_n)c_1 \wedge \cdots c_m$ be a quantified 3CNF formula with m clauses over n variables, and let a be the number of its universal quantifiers. The graph G has $3m$ cliques of $2n$ nodes each, where each possible literal corresponds to one node in each clique. The cliques are grouped into m clusters of three cliques. Each clause c_i corresponds to a cluster of cliques C_{i1}, C_{i2}, C_{i3} . For each clause $c_i = (l_{i1} \vee l_{i2} \vee l_{i3})$, a triangle of edges connects the cliques of its cluster: from l_{i1} in C_{i1} to l_{i2} in C_{i2} , from l_{i2} in C_{i2} to l_{i3} in C_{i3} , and from l_{i3} in C_{i3} to l_{i1} in C_{i1} .

There are $m(3n + 1)$ guards. The (unnatural) setting requires the guards to move as follows. In the first round, the guards must clear cliques C_{i1}, C_{i2} , $1 \leq i \leq m$, and all edges between cliques. Over the next 2^{a+1} rounds, the guards must walk through a series of 2^a “assignments.” For each assignment, n of the nodes on each clique must be occupied, corresponding to the true literals for that assignment. The remaining m guards can occupy only nodes of clear cliques, and at most one extra guard per cluster.

The series of variable assignments must be as follows. Let A be the set of universally quantified variables and E the set of existentially quantified variables in the formula. The first variable assignment must set every variable in A to zero, and has arbitrary values for every variable in E . Let ϕ_k be the k th variable assignment, and suppose that i is the largest index such that $x_i \in A$, $\phi_k(x_i) = 0$. Then ϕ_{k+1} must satisfy $\phi_{k+1}(x_i) = 1$; $\phi_{k+1}(x_j) = 0$ for all $x_j \in A$ such that $j > i$; $\phi_{k+1}(x_j) = \phi_k(x_j)$ for all $j < i$; and $\phi_{k+1}(x_j)$ arbitrary for all $x_j \in E$ such that $j > i$.

The guards must walk through these assignments in order $\phi_1, \dots, \phi_{2^a}$, with one round after each assignment as follows. After simulating ϕ_k in round $2k$, one guard from each C_{i3} is allowed to move in round $2k + 1$ to occupy the node corresponding to l_{i3} . (This clears all the triangles if the search has been successful so far.)

After these assignments are simulated, the guards are allowed a single round (round $2^{a+1} + 2$) to fully clear cliques C_{i3} , $1 \leq i \leq m$, and all edges between cliques. Notice that this is a legitimate setting, since it is oblivious, and efficient (e.g., the validity of the next assignment can be determined from the previous assignment in polynomial time).

No search can succeed if one of its assignments leaves all three vertices of a triangle unoccupied (since one extra guard is not enough to prevent recontamination of one of the two clear cliques). On the other hand, a search can succeed if none of its assignment leaves all three vertices of any triangle unoccupied (since the extra guards can always ensure that no recontamination occurs). Thus the graph can be cleared in this setting with $m(3n + 1)$ guards if and only if there is an appropriate series of variable assignments that all satisfy the 3CNF formula, i.e., if and only if the quantified formula is satisfiable. \square

The proof of this claim relies on the use of a mobility setting with unusual, interrelated constraints: (1) allowing restrictions on the location of a guard to be dependent on the round number; (2) forbidding certain subsets of nodes from being partially occupied (i.e., neither full nor vacant) during certain rounds. The complexity of the decision problem for more “natural” settings (where

restrictions do not vary over time or interrelate guards) are considered in a later section.

4.5 Database Maintenance: Separations and Complexity

In this section, we consider the sensitivity of distributed database maintenance to the choice from among a family of simple mobility settings.

4.5.1 Definition of Specific Settings

Among the unlimited possibilities for constraining the movement of bugs (guards), we identify some interesting and natural settings by considering all combinations of the following options:

- **Frequent/Infrequent:** Either the adversary can change bug (guard) locations at the end of every round (frequent), or only at the end of every $|V|$ th round (infrequent).
- **Parallel/Sequential:** When bugs (guards) can move, either the adversary can move any number (parallel), or at most one (sequential).
- **Hopping/Crawling:** When a bug (guard) moves, either the adversary can relocate it at any node in the graph (hopping), or only at a node adjacent to its current location (crawling).

We also consider the “location-restricted” versions of these mobility settings. For this type of setting, a subset of nodes is associated with each bug (guard), and every bug (guard) must stay within its subset throughout the game.

We may abbreviate settings by using initials (e.g., F/P/H for frequent parallel hopping guards), possibly with wild cards (e.g., I/*/* for any setting with infrequent movement). Notice that these settings form a lattice with respect to adversarial power. For example, the adversary is at least as strong if Frequent rather than Infrequent, if Parallel rather than Sequential, or if Hopping rather than Crawling. As we shall see, some of the other pairs of settings are incomparable with respect to node searching a graph.

However, we can show that when bug (guard) movement is infrequent, the settings are equivalent for the problem of distributed database maintenance. Having the guards move once every $|V|$ rounds is equivalent to allowing the contamination to spread arbitrarily fast every round, i.e., the original node search model due to Kirousis and Papadimitriou [100]. The following theorem can then be proven from the result of LaPaugh [104] that “recontamination doesn’t help” in these settings, i.e., for any successful search there is a search without recontamination using the same number of guards (proof omitted).

Theorem 4.4 *For any graph G , $ns_{I/P/C}(G) = ns_{I/P/H}(G) = ns_{I/S/C}(G) = ns_{I/S/H}(G)$.*

The node search number of a graph versus arbitrarily fast recontamination (I/*/* setting) has been shown to be connected to other topological properties of the graph (e.g., vertex separator [100], interval thickness [99], topological bandwidth [108], cutwidth [108]). No similar results are known for any F/*/* settings.

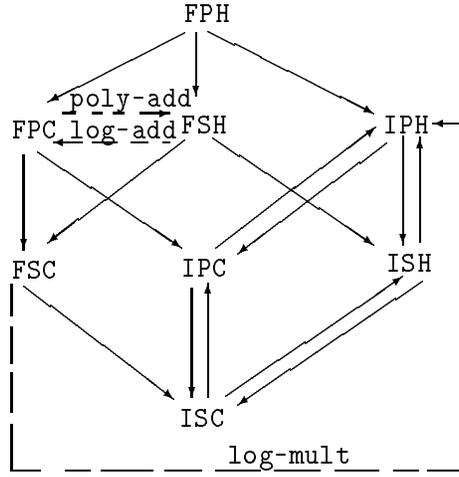


Figure 4.3: Lattice of Settings with Separation Results

4.5.2 Separation among Settings

In this section, we show that the ability to maintain a distributed database can depend significantly on the mobility setting, by showing gaps between the search numbers of graphs in different I/*/* and F/*/* settings.

Let σ_1, σ_2 be settings. We say that σ_1 “can beat” σ_2 if there exists a graph G such that $ns_{\sigma_1}(G) < ns_{\sigma_2}(G)$. We say that σ_1 can beat σ_2 “ f -additively” (“ f -multiplicatively”) if for every sufficiently large n there exists a graph G_n of size $poly(n)$ such that $ns_{\sigma_1}(G_n) < ns_{\sigma_2}(G_n) + \Theta(f(n))$ (such that $ns_{\sigma_1}(G_n) < ns_{\sigma_2}(G_n) * \Theta(f(n))$).

The results of this section are summarized in Figure 4.3, which shows the lattice of settings and the separations proven in Theorems 4.5, 4.6, and 4.7. In the figure, a solid arrow from σ to σ' means that $ns_{\sigma}(G) \leq ns_{\sigma'}(G)$ for all G , double arrows indicate the equivalences proven in Theorem 4.4, and dashed arrows indicate the separations.

Theorem 4.5 *Any F/*/*/H setting can beat any */*/C setting log-additively.*

Proof: It suffices to show that there is a family of poly-sized graphs $\{G_n\}$ such that $ns_{F/S/H}(G_n) \leq n + 2$ while $ns_{F/P/C}(G) \geq n + \log n$. See Figure 4.4.

Let $k = \lceil \log n \rceil$. G_n consists of 4 complete subgraphs $\{L_1, \dots, L_n\}$, $\{T_1, \dots, T_n\}$, $\{B_1, \dots, B_n\}$, $\{R_1, \dots, R_n\}$. There are connecting edges (L_i, T_i) , (T_i, B_i) , for every i , $1 \leq i \leq n$. There are connecting paths from T_j to R_j for every j , $1 \leq j \leq k$, of varying lengths to be determined later.

To search G_n in the F/S/H setting with $n + 2$ guards, the guards clear L and then move so as to clear T without recontaminating L . Then the two free guards clear the first $k - 2$ paths from T to R , where the path from T_i to R_i is cleared to distance d_i . After the two free guards return to T_{k-1}, T_k , the n original guards on T move to B , clearing it. Then the guards at B return to T , arriving in the order $T_{k-2}, T_{k-3}, \dots, T_1, T_{k+1}, \dots, T_n$. Lastly, the guards clear along the paths to R

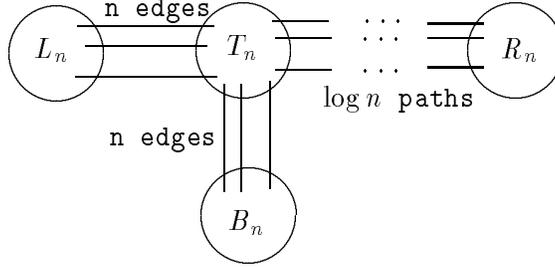


Figure 4.4: F/*/H Setting Beats */*/C Setting Log-Additively

without recontamination, completing the search of the graph. There exists a small constant c such that the search will be successful if $d_{k-2} = n + 1$ and $d_{k-i} = 2d_{k-i+1} + c$ for every $3 \leq i \leq k - 1$ (i.e., T will never be recontaminated). We can now say how long the paths from T to R need to be: It suffices for the path from T_j to R_j to be of length d_j for every j , $1 \leq j \leq k - 2$, while the paths from T_{k-1} to R_{k-1} and from T_k to R_k can be a single edge.

We show that G_n cannot be cleared in the F/P/C setting with $n + k - 1$ guards. Consider the order in which the four complete subgraphs L, T, B, R are permanently cleared. It can be shown that, for every order, the third subgraph cannot be cleared without recontaminating one or both of the first two. We show this for the most difficult case, in which the order of permanent clearance is L, T, B, R .

Suppose that B is permanently cleared during some round i . This recontaminates T unless all k paths are cleared to distance at least one at the start of round i . However, the following round invariant holds. If λ paths are cleared to distance one or more, then at least λ guards are also this far from T . Since B is cleared during round i , some guard moves from T to B (or within B) during round i leaving at most $k - 1$ guards outside of B . By the invariant, some path is both uncleared and unoccupied at T during round i , and thus T is recontaminated by the end of round i . \square

Theorem 4.6 *Any F/P/* setting can beat any */S/* setting poly-additively.*

For the proof of this theorem, we need the following lemma about possible patterns of contamination in a graph.

Lemma 4.1 *If G' is a complete subgraph of G , then at the end of every round i one of the following is true: (1) G' is fully clear; (2) G' was fully clear at the end of round $i - 1$, and for one or more unoccupied nodes on the boundary of G' and $G - G'$, every edge of G' adjacent to those nodes is contaminated; or (3) every edge adjacent to an unoccupied node is contaminated.*

Proof : This is clearly true at the end of the first round, when the only clear edges are those occupied at both endpoints. Suppose it is true through the end of round i . If the first condition is satisfied at the end of round i , then the second condition is trivially satisfied at the end of round $i + 1$. If the third condition but not the first is satisfied at the end of round i , then all guard

additions, deletions, and movements will either fully clear G' or preserve the third condition. If the second condition but not the first is satisfied at the end of round i , then all edges adjacent to one or more unoccupied nodes of G' are contaminated at the end of round i , and this contamination will spread to all edges adjacent to unoccupied nodes at the end of round $i + 1$ for any possible movement of guards. \square

Proof: (Theorem) It suffices to show that there is a family of poly-sized graphs $\{G_n\}$, $n \geq 2$, such that $ns_{F/P/C}(G_n) \leq 3n$ while $ns_{F/S/H}(G) \geq 4n - 1$. See Figure 4.5.

G consists of 3 complete subgraphs on $3n$ vertices $\{L_1, \dots, L_{3n}\}$, $\{B_1, \dots, B_{3n}\}$, $\{R_1, \dots, R_{3n}\}$. There are also connecting edges (L_i, B_i) , (L_i, R_i) , (R_i, B_{n+i}) for every i , $1 \leq i \leq n$. The search in the F/P/C setting with $3n$ guards is as follows. The guards clear L by occupying its $3n$ vertices. Then the guards move so as to occupy $\{L_1, \dots, L_n, B_1, \dots, B_n, R_1, \dots, R_n\}$ while clearing all connecting edges to L , without recontaminating L (two rounds). Then the n guards on L move to B_{n+1}, \dots, B_{2n} (in two rounds), clearing all edges between R and B without any recontamination. Then B is cleared in a single round, as the n guards on B_{n+1}, \dots, B_{2n} move to B_{2n+1}, \dots, B_{3n} simultaneously with the n guards at R moving to B_{n+1}, \dots, B_{2n} (recontaminating the LR edges). Then the guards at B_1, \dots, B_n and B_{2n+1}, \dots, B_{3n} return to L_1, \dots, L_n (one round for the first set of guards, and two rounds for the second set). Then n of the guards on L move to R_1, \dots, R_n , clearing all edges to R . The rest of R is cleared two rounds later, as the remaining guards from L and B move to the remaining locations on R .

We claim that this graph cannot be searched with $4n - 2$ guards in the F/S/H setting (although, in fact, $4n - 1$ guards suffice). Assume that some search in the F/S/H setting is successful with only $4n - 2$ guards. There must be an earliest round i at the end of which two of the complete subgraphs L, B, R are cleared. Suppose wlog that B can be cleared in round i while L remains clear. Thus there must be a round $j < i - 1$ such that L is clear at the start of round j , $3n - 2$ guards occupy B at the start of round j , a $(3n - 1)$ st guard enters B during round j , and these $3n - 1$ guards remain in B until the end of round $i > j + 1$. Since i is the earliest such round, we must have that R is at least partly contaminated at the start of rounds j and $j - 1$. This implies that every edge adjacent to an unoccupied node in R is contaminated at the start of round j (by Lemma 4.1). The $n - 1$ guards outside B cannot prevent the recontamination of L from along one of the n edges from L to R . \square

Theorem 4.7 *Any F/*/* setting can beat any I/*/* setting log-multiplicatively.*

Proof: It suffices to show that there is a family of poly-sized graphs $\{G_n\}$ such that $ns_{F/S/C}(G_n) \leq 4$ while $ns_{I/*/*}(G_n) = \Omega(\log n)$. See Figure 4.6.

Each graph G_n has a subgraph that is homeomorphic to a full ternary tree $T_{\lceil \log n \rceil}$, defined as follows. T_0 is a ternary tree with seven nodes, where the root has three children, and where each of these children has a single child. T_k is a ternary tree with three copies of T_{k-1} connected to paths from the root of varying lengths. In addition to this subgraph, each G_n has a special “hub” node, with edges to every node of the tree. In the F/S/C setting, a guard can move from any node to any other node in at most two rounds (through the hub).

The search procedure for G_n with four guards is as follows. One guard remains on the hub throughout the search. The other three guards start at the root of the tree. Two guards clear the right path, return to the root via the hub, clear the middle path, return to the root via the hub,

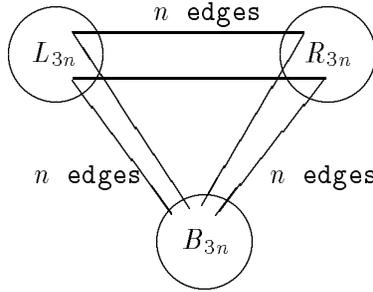


Figure 4.5: F/P/* Setting Beats */S/* Setting Poly-Additively

and then clear the left path. Now the third guard, via the hub, joins the other two at the root of the left subtree. These three guards clear the left subtree recursively. When the left subtree is cleared, the three guards return to the root of the tree via the hub. Then two guards clear the right path, return to the root via the hub, and clear the middle path. Now the third guard, via the hub, joins the other two at the root of the middle subtree. These three guards clear the middle subtree recursively. When the middle subtree is cleared, the three guards return to the root of the tree via the hub. Then two guards clear the right path, and are joined, via the hub, at the root of the right subtree by the third guard. Lastly, the three guards clear the right subtree recursively.

Let t_k denote the number of steps required to clear T_k by three guards (with a fourth guard occupying a hub). Let l_k, m_k, r_k denote the lengths of the left path, middle path, and right path of T_k . It takes eleven rounds for three guards to clear T_0 (starting at the root, and using the hub), i.e., $t_0 = 11$. There exists small constants c_1, c_2 such that the search of T_k plus hub will be successful if we have $l_k = 1$, $m_k = t_{k-1} + c_1$, and $r_k = 3t_{k-1} + c_2$. Since $k = O(\log n)$, these recurrences imply that the size (and search time) of G_n is polynomial in n .

A full ternary tree of height h requires $h + 1$ guards to node search in any I/** setting (by a theorem of Kirousis and Papadimitriou [100]). Since adding edges to a graph cannot decrease the search number in an I/** setting, each G_n requires $\Omega(\log n)$ guards to search in any I/** setting. \square

For the family of graphs used to prove Theorem 4.5, success depends critically on the length of paths between subgraphs. For the family of graphs used to prove Theorem 4.6, success in the F/S/C setting depends critically on the number of edges connecting subgraphs. This contrasts with the I/** settings, where “stretching” or adding edges can never decrease the search number of a graph.

4.5.3 Search in a Frequent Setting Can Require Exponential Time

In this section, we indicate another way that searching versus a slow fugitive (Frequent settings) can be quite different from searching versus a fast fugitive (Infrequent setting). The theorem of LaPaugh [104] shows that, in an I/** setting, any graph $G = (V, E)$ can be searched by $ns_{I/**}(G)$ guards in $\text{poly}(|V|)$ steps. By contrast, we can prove that a search in a Frequent setting can be

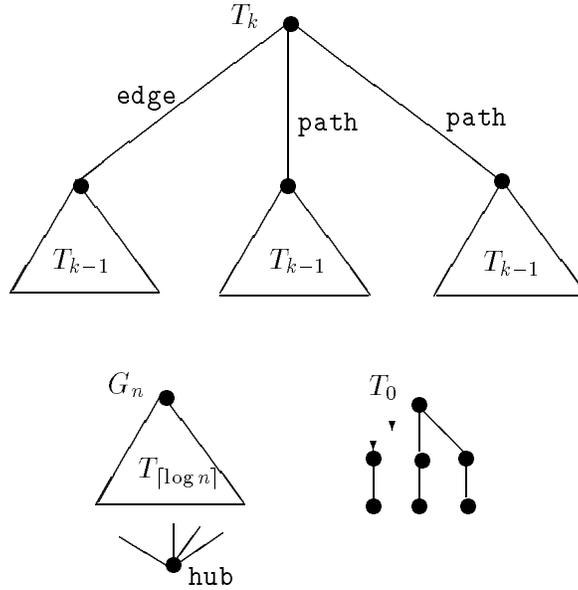


Figure 4.6: F/** Setting Beats I/** Setting Log-Multiplicatively

necessarily exponential in length.

Theorem 4.8 *There exists a graph $G = (V, E)$ with $|V| = O(n^2)$ such that the shortest search with n guards in the F/P/H setting takes $\Omega(2^n)$ steps.*

The following lemma about “purposeful” searches will be useful. Define the “state” of a graph at any moment to be given by the set of complete subgraphs (cliques) that are fully clear at that moment. We call a search “purposeful” if (a) no two rounds begin in the same state (up to symmetry); (b) every guard move helps either to fully clear a clique or to prevent immediate recontamination of a fully cleared clique; and (c) at least one clique becomes fully cleared every round.

Lemma 4.2 *The shortest search of any graph in the F/P/H setting is purposeful.*

Proof: One consequence of Lemma 1 is that every node in a partly contaminated clique is adjacent to a contaminated edge. Another consequence of Lemma 4.1 is that a partly contaminated clique cannot be fully cleared until all nodes are occupied simultaneously. Thus for purposes of graph searching there is no difference between a fully contaminated clique and a partly contaminated clique. Any movement of guards on a partly contaminated clique that doesn’t fully clear it in that round can be postponed until the round when the clique is fully cleared without affecting the success of the search.

Similarly, any movement of guards on a clear clique that doesn’t immediately prevent recontamination can be delayed either indefinitely or until recontamination is immediately threatened, without affecting the success of the search.

If a state of the graph is repeated (up to symmetry), then all rounds after the first occurrence of the state and up to the second occurrence can be eliminated without affecting the success of the search (after appropriately remapping all guard movements before the first occurrence if the repetition is only up to symmetry).

If a round does not fully clear a clique, then its guard movements can be combined with the subsequent round without affecting the success of the search. \square

Proof : (Theorem) The graph G consists of a (left) complete graph L_n on n vertices x_1, \dots, x_n , a (right) complete graph R_n on n vertices y_1, \dots, y_n , and (middle) complete graphs M_i on i vertices such that $x_i, y_i \in M_i$ for every i , $2 \leq i \leq n$. See Figure 4.7.

This graph is searchable with n guards. To help describe the search, we will say that G is “cleared to M_i ” if at the end of some round there are guards at $x_2, y_2, \dots, y_i, x_{i+1}, \dots, x_n$, while L_n, M_2, \dots, M_i are fully cleared. The n guards begin at x_1, \dots, x_n , clearing L_n . Then the guard at x_1 moves to y_2 , and at the end of this round the graph is cleared to M_2 . Notice that the guards above M_2 did not have to move to clear G to M_2 . Suppose that the graph is cleared to M_i at the end of round $r + 2$. During the next round $r + 3$, the guards at x_2, y_2, \dots, y_i move to M_{i+1} , clearing M_{i+1} while recontaminating M_2, \dots, M_i . During the next round $r + 4$, the guard at y_{i+1} doesn’t move, while the other guards in M_{i+1} move to y_2, x_2, \dots, x_i . Over the next r rounds, the same movement of guards that originally cleared the graph to M_i is repeated; by the end of round $2r + 4$ the graph is cleared to M_{i+1} . When the graph has been cleared to M_n (with guards at x_2, y_2, \dots, y_n), the guard at x_2 moves to y_1 to complete the search. The total search time is $2^{n-1} + 1$ rounds.

We now show that n guards cannot search this graph in fewer rounds. By Lemma 4.2, we can restrict our attention to purposeful searches of G .

At some point in the search of G , either L_n or R_n will be cleared for the first time. At the end of that round, all other cliques will be partly contaminated. We can assume, then, that one of L_n or R_n is cleared in the first round; wlog L_n is cleared in the first round. Since the search is purposeful, R_n is not cleared until the final round (else the state after clearing R_n is the same up to symmetry as the state after the first round), and L_n is never recontaminated (else the state after reclearing L_n is the same as the state after the first round).

Call the graph “clear at m ” if the fully clear cliques are $L_n, M_{i_1}, \dots, M_{i_k}$, where $m = 2^{i_1-2} + \dots + 2^{i_k-2}$. We need a purposeful search of G with n guards that is clear at 0 after the first round, and clear at $2^{n-1} - 1$ after the penultimate round.

We claim that the only possible purposeful search of this kind has the following structure. Let $I_m = \{i_1, \dots, i_k\}$ where $m = 2^{i_1-2} + \dots + 2^{i_k-2}$, $2 \leq i_1 < \dots < i_k \leq n$. Then for every $0 \leq m \leq 2^{n-1} - 1$: (a) G is clear at m at the end of round $m + 1$; (b) M_{i_1} was the only clique to become clear during round $m + 1$; (c) M_2, \dots, M_{i_1-1} became contaminated during round $m + 1$; and (d) guards occupy $M_{i_1}, y_{i_2}, \dots, y_{i_k}, \{x_j : j > i_1, j \notin I_m\}$ at then end of round $m + 1$.

This claim is vacuously satisfied at the end of the first round. Suppose it is true through the end of round $m + 1$. Let $I_m = \{i_1, \dots, i_k\}$.

If $i_1 > 2$, then guards must move during round $m + 2$ to occupy x_2, \dots, x_{i_1-1} (else L_n is recontaminated). No guard above M_{i_1} can move purposefully in round $m + 2$ without recontaminating L_n or repeating a state, so the only clique that can be cleared by available guards is M_2 . [Let $I_m^* \subseteq I_m$ be the indices of the cliques that get recontaminated in round $m + 2$, and let $J_m^* \subseteq J_m$ be the indices of cliques that get cleared in round $m + 2$. Since guards must move to x_2, \dots, x_{i_1-1} ,

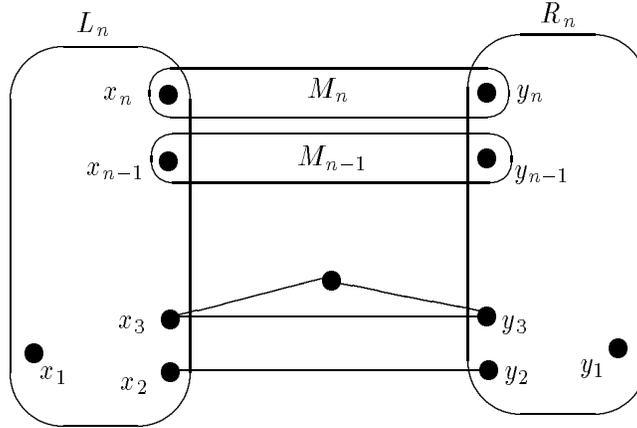


Figure 4.7: Search in F/P/H Setting Can Require Exponential Rounds

and since no guard can move from $\{x_j : j > i_1, j \notin I_m\}$, to prevent recontamination of L_n , at most $|I_m^*| - i_1 + 2$ guards are available to clear the cliques in J_m^* . Just to clear $M_{\max J_m^*}$ requires that $|I_m^*| - i_1 + 2 \geq \max J_m^* - 1$. But this implies that $\max I_m^* > \max J_m^*$, which means that state of being clear at $m - \sum_{i \in I_m^*} 2^{i-2} + \sum_{j \in J_m^*} 2^{j-2}$ gets repeated.]

If $i_1 = 2$, then let $j = \min \bar{I}_m$. No guard above M_j can move purposefully in round $m+2$ without recontaminating L_n or repeating a state. Thus the only clique that can be cleared by available guards is M_j , using the guards at M_2, y_3, \dots, y_{j-1} . This recontaminates M_2, M_3, \dots, M_{j-1} .

In either case, the four conditions are met at the end of round $m+2$. Since this is the only possible purposeful search, it will take 2^{n-1} rounds to be clear at $2^{n-1} - 1$, and the entire search will take $2^{n-1} + 1$ rounds. Since no purposeful search can be shorter, no search of G can be done in fewer rounds. \square

Similar examples are possible in the other F/*/* settings.

4.5.4 Complexity of Node Search Problems

Given a graph G , a number K , and a mobility setting σ , the node search decision problem determines whether $ns_\sigma(G) \leq K$. We know that this problem is in general PSPACE-complete, but it is interesting to consider the problem for natural settings. A complete answer can be given for Infrequent settings, and a partial answer for Frequent settings.

By the NP-completeness of node searching as defined by Kirousis and Papadimitriou [100] [104] [110], together with Theorems 4.3 and 4.4, we can show the following claim (proof omitted).

Claim 4.5 *The node search decision problem in an I/*/* setting is NP-complete.*

For the Frequent settings, membership in NP is possible, but unlike the I/*/* settings, the search itself cannot serve as a witness for membership (see Theorem 4.8). NP-hardness can be shown.

Theorem 4.9 *The node search decision problem in an F/*/* setting is NP-hard.*

Proof : We show that determining the node search number in these settings is NP-hard, by modifying the proof of NP-hardness for edge searching versus a fast fugitive due to Megiddo *et al* [110]. Let $G = (V, E)$ and $K > 0$ be an instance of MIN-CUT INTO EQUAL-SIZED SUBSETS (i.e., determine whether V can be partitioned into equal sized subsets such that at most K edges of E are cut). We construct a corresponding node search problem as follows. Let $n = |V|$, let d be the maximum vertex degree in G , let $N = 6(d + K)$, and let $M = n(n + 2)N$.

The graph to be searched is $H = (U, F)$ defined as follows. For each vertex $v_i \in V$, there is an M -clique C_i . There is an additional “special” M -clique C_A . Furthermore, every pair of cliques C_i, C_j shares some number of vertices: $nN + N$ vertices if either i or j is A ; $nN + 3$ vertices if $(v_i, v_j) \in E$; nN vertices otherwise. This is done in such a way that no vertex belongs to more than two cliques, which is possible since M is sufficiently large. [Note that this is the same construction as in [110] except that all edges between cliques have been contracted.]

Call a clique unclear if it has at least one uncleared edge, and clear otherwise. Recall from Lemma 4.1 that every node in an unclear clique is adjacent to at least one uncleared clique edge.

If there is a min-cut partition of G into $V_1 = \{v_1, \dots, v_{\frac{n}{2}}\}$ and $V_2 = \{v_{\frac{n}{2}+1}, \dots, v_n\}$, then H can be node searched in any setting with $s = M + (\frac{n}{2})^2 nN + 3K$ guards. The cliques are cleared in the order C_1, \dots, C_n , with C_A cleared after $C_{\frac{n}{2}}$ and before $C_{\frac{n}{2}+1}$, with guards moving so that no recontamination occurs. It can be shown that s guards suffices: If the clique being cleared is not C_A then the number of guards needed is at most $M + (\frac{n}{2} - 1)(\frac{n}{2} + 1)nN + (\frac{n}{2} - 1)(N + 3d) < s$; and if the clique being cleared is C_A then the number of guards needed is at most $M + (\frac{n}{2})^2 nN + 3K = s$.

[A little more detail for this last sentence: If $\frac{n}{2} - i, i > 0$, cliques are already cleared, then at most M guards are needed for clique $C_{\frac{n}{2}-i+1}$, while at most $(\frac{n}{2} - i)(\frac{n}{2} + i)nN + (\frac{n}{2} - i)N + (\frac{n}{2} - i)3d$ guards are needed to cover the default common vertices, the extra common vertices with C_A , and the extra common vertices due to edges in E ; and this is maximized when $i = 1$. If $\frac{n}{2} + i, i > 0$, cliques are already cleared, the count is identical. If exactly $\frac{n}{2}$ cliques have been cleared, and the next clique to be cleared is C_A , then $(\frac{n}{2})^2 nN$ guards are needed to cover the default common vertices, at most M guards are needed to cover the nodes of C_A , and at most $3K$ guards are needed to cover the extra common vertices due to edges in E .]

For the other direction, suppose that there is a search strategy for H with s guards in some F/*/* setting; then there is a search strategy with s guards in the F/P/H setting. Consider the round when H first reaches or surpasses $\frac{n}{2} + 1$ clear cliques. Before this round there were $\frac{n}{2} + 1 - j$ clear cliques and $\frac{n}{2} + j$ unclear cliques for some $j > 0$. During this round i cliques go from unclear to clear, while $i - j - l$ cliques go from clear to unclear, for some $i > 0, l \geq 0$. Thus $\frac{n}{2} - i + l + 1$ remain clear and $\frac{n}{2} - i + j$ remain unclear during this round. At the end of this round, there are $\frac{n}{2} + l + 1$ clear and $\frac{n}{2} - l$ unclear cliques. During this time step, the following must be true:

1. Every vertex shared by a clique that stays clear and a clique that stays unclear must be occupied by a guard.
2. Every vertex on a clique that becomes clear must be occupied by a guard.

A total of iM guards are needed for the second requirement. At least $(\frac{n}{2} - i + l + 1)(\frac{n}{2} - i + j)nN$ guards are needed for the first requirement (not counting the extra three or N vertices that two

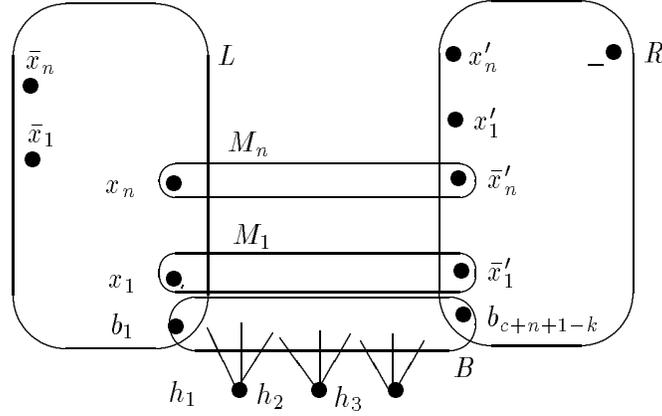


Figure 4.8: Location-Restricted F/P/H Setting is co-NP-hard (partial view)

cliques might share). Adding just these gives $iM + (\frac{n}{2} - i + l + 1)(\frac{n}{2} - i + j)nN$ guards needed. This is already larger than s when $i > 1$, so we only need to consider $i = 1$, i.e., one clique goes from unclear to clear.

If the clique that goes from unclear to clear is not C_A , then we need extra guards on the common vertices between C_A and cliques of opposite type from C_A (i.e., unclear cliques if C_A is clear, or clear cliques if C_A is unclear). In either case this is at least $\frac{n}{2}N$ additional guards, for a total of more than s guards for every $j > 0, l \geq 0$.

If the clique to be cleared is C_A , then we need extra guards on common vertices between cleared normal cliques and uncleared normal cliques. If y is the number of edges between the two sets of corresponding nodes in the original graph G , then $3y$ is the number of common vertices needing additional guards. This gives a total number of guards needed which is more than s unless $j = 1, l = 0$, and $3y \leq 3K$. In this case, we have found a min-cut, i.e., $y \leq K$. \square

If bugs in the F/P/H mobility setting are restricted to (possibly overlapping) subsets of nodes, then the decision problem is co-NP-hard.

Theorem 4.10 *The node search decision problem in the location-restricted F/P/H setting is co-NP-hard.*

Proof: Our proof is by reduction from the complement of MIN-2-SAT. MIN-2-SAT (given a 2-CNF formula in n variables, is there an assignment that satisfies less than k clauses?) is NP-complete, by reduction from MAX-2-SAT: Let ψ, k be an instance of MAX-2-SAT, where ψ has c clauses. For each 2-clause $(l_i \vee l_j)$ in ψ , put clauses $(\bar{l}_i \vee l_j), (l_i \vee \bar{l}_j), (\bar{l}_i), (\bar{l}_j)$ in ψ^* . For each 1-clause (l_i) in ψ , put clauses $(l_i), (l_i), (\bar{l}_i), (\bar{l}_i), (\bar{l}_i), (\bar{l}_i)$ in ψ^* . Then an assignment satisfies more than k clauses of ψ if and only if that assignment satisfies less than $4c - 2k$ clauses in ψ^* .

Let $\psi = w_1 \vee \dots \vee w_c$ be a 2-CNF formula on n variables with c clauses, and let $c - k > 0$. We construct a graph G and a restricted range mobility setting such that G is searchable by $3n + c + 2$

guards if and only if $\psi, c - k$ is in co-MIN-2-SAT (i.e., iff every assignment satisfies at least $c - k$ clauses). See Figure 4.8 for a partial view of G .

There is a left clique L with nodes $x_1, \dots, x_n, \bar{x}_1, \dots, \bar{x}_n, z_1, \dots, z_n, b_1$. There is a right clique R with nodes $x'_1, \dots, x'_n, \bar{x}'_1, \bar{x}'_n, z'_1, \dots, z'_n, b_{c+n+1-k}, -$. There is a bottom clique B with nodes $b_1, \dots, b_{c+n+1-k}$ and “horns” h_1, h_2, h_3 , where an edge connects each h_i to each b_j . There are middle cliques M_1, \dots, M_n , where each M_i has nodes x_i, \bar{x}'_i, m_i^* , and m_{ij} for each $1 \leq j \leq n + i - 1$, $j \neq i$. There is a (“default”) path of length two between \bar{x}_i and x'_i with midpoint δ_i , for each $1 \leq i \leq n$. There is a (“clause”) path of length two between l_i and l'_j (or between l_i and $-$) with midpoint ϵ_λ for every 2-clause $w_\lambda = (l_i \vee l_j)$ (or 1-clause $w_\lambda = (l_i)$) in ψ , $1 \leq \lambda \leq c$.

One guard g^h is restricted to $h_1, h_2, h_3, -$. One guard g^* is restricted to $b_1, b_{c-k+n+1}$ and m_i^* for every $1 \leq i \leq n$. For each i , $1 \leq i \leq n$, there is one guard g_i restricted to x_i, \bar{x}_i, z'_i , and $m_{j,n+i}$ for every $j > i$. For each i , $1 \leq i \leq n$, there is one guard g'_i restricted to x'_i, \bar{x}'_i, z_i , and m_{ji} for every $j \neq i$. For each i , $1 \leq i \leq n$, there is one guard g_i^δ restricted to $\delta_i, \bar{x}_i, x'_i, b_{c-k+i+1}$. For each λ , $1 \leq \lambda \leq c$, there is one guard g_λ^ϵ restricted to ϵ_λ and b_j for every $2 \leq j \leq c - k + 1$.

For one direction, we show that G is searchable when every assignment satisfies at least $c - k$ clauses of ψ . We will say that G is clear “at v ” when (1) $v = 2^{i_1} + \dots + 2^{i_m}$, $1 \leq i_1 < \dots < i_m \leq n$; (2) for each $1 \leq i \leq n$, guards are at x_i, x'_i if $i \neq i_1, \dots, i_m$ and at \bar{x}_i, \bar{x}'_i otherwise; (3) cliques $L, M_{i_1}, \dots, M_{i_m}$ are fully clear; and (4) the edges connecting L to every default path and clause path are clear. The assignment “at v ” assigns false to variables x_{i_1}, \dots, x_{i_m} (and corresponding variables $x'_{i_1}, \dots, x'_{i_m}$) and true to all others. When describing the search, we usually indicate only the guards that move.

The search begins by clearing the default paths edges adjacent to L : $\forall i g_i$ at \bar{x}_i , g'_i at \bar{x}'_i , g_i^δ at δ_i ; $\forall \lambda g_\lambda^\epsilon$ at ϵ_λ ; g^* at b_1 ; g^h at $-$. Then L is cleared: $\forall i g_i$ at x_i , g'_i at z_i , g_i^δ at \bar{x}_i . Then one more round brings the graph to being cleared at zero: $\forall i g'_i$ at x'_i , g_i^δ at δ_i .

It takes six rounds for G to go from being cleared at v to being cleared at $v + 1$, as follows. Let $v = 2^{i_1} + \dots + 2^{i_m}$, $1 \leq i_1 < \dots < i_m \leq n$, where j is the smallest missing exponent. First B is cleared in three rounds: $\forall i g_i^\delta$ at $b_{c-k+i+1}$, g_λ^ϵ at b_p for $2 \leq p \leq c - k + 1$ (where clause w_λ is satisfied by the assignment “at v ” for every guard g_λ^ϵ that moved to B); g^* at b_1 ; and g^h goes to h_1 and then h_2 and then h_3 . Next we prepare to advance from v to $v + 1$: $\forall i g_i^\delta$ at δ_i , $\forall \lambda g_\lambda^\epsilon$ at ϵ_λ , g^* at $b_{c-k+n+1}$; g^h at $-$. Then we clear the lowest unclear middle clique M_j : $\forall i \neq j g'_i$ at m_{ji} ; $\forall i < j g_i$ at $m_{j,n+i}$; g^* at m_i^* . Lastly we move to being at $v + 1$: $\forall i < j g_i$ at x_i , g'_i at x'_i ; g_j at \bar{x}_j ; g'_j at \bar{x}'_j ; g^* at b_1 ; $\forall i > j g'_i$ reflects location of g_i (i.e., at x'_i if $i \neq i_1, \dots, i_m$ and at \bar{x}'_i otherwise).

When prepared to take the last step to advance from being at $2^{n+1} - 1$ to 2^{n+1} , the edges connecting all default and clause paths to R are cleared: $\forall i g'_i$ at x'_i . Then the clique R is cleared to complete the search: $\forall i g'_i$ at \bar{x}'_i , g_i^δ at x'_i , g_i at z'_i .

If every assignment satisfies at least $c - k$ clauses, then $c - k$ guards can always be chosen to help clear B (i.e., so that L is not recontaminated through R along some clause path during the three steps required to clear B).

For the reverse direction, we need to show that if some assignment satisfies less than $c - k$ clauses, then G cannot be searched successfully. Consider the relative order in which L, R and all of M are permanently cleared. R will be contaminated after any M clique except the last is cleared; thus R cannot be permanently cleared first. If L is cleared while R is partly contaminated, or if R is cleared while L is partly contaminated, then all M cliques become contaminated; thus M cannot be permanently cleared first. If R (or L) is cleared while some M clique is partly contaminated,

then L (or R) becomes contaminated; thus M cannot be permanently cleared last.

The only possible ordering is L first, M second, and R third. After L is permanently cleared, a progress argument similar to the proof of Theorem 4.8 shows that every assignment “at v ” must be visited, for every $0 \leq v \leq 2^{n+1} - 1$, without recontaminating L . To advance from v to $v + 1$ requires that all but k clause paths be unguarded in the middle for three rounds (to free the guard g^* for one round). If some assignment does not satisfy at least $c - k$ clauses, then some length two clause path connecting permanently cleared L to contaminated R will be unguarded at the middle and at both ends for at least three rounds, a contradiction. \square

Although we don’t yet know the exact complexity of the decision problem in the unrestricted F/P/H settings, notice that we get an interesting complexity separation in any case. If unrestricted F/P/H is in NP, then Theorem 4.10 implies a separation between unrestricted and restricted F/P/H (unless $\text{NP} = \text{co-NP}$). If unrestricted F/P/H is harder than NP, then Claim 4.4 implies a separation between F/P/H and I/P/H.

4.6 Open Problems

We would like to know the exact complexity of determining the node search number of a graph in other natural mobility settings, with particular interest in the unrestricted F/*/* settings. We would also like to prove lower bounds on resources needed to win various eavesdropping games. It may also be interesting to consider graph-theoretic analyses of security problems versus stronger adversaries (e.g., to introduce faults).

An intriguing direction of research is to investigate the effects of relaxing the synchrony constraints in the model of eavesdropping games. Some preliminary results have already been obtained along these lines. For example, consider the “unsynchronized” model in which bug movement and network behavior are each synchronous, but not synchronized with one another. For any eavesdropping game, if a network can defeat $2t$ bugs controlled by the strongest (F/P/H) *synchronized* adversary, then the network can defeat t bugs controlled by the strongest (F/P/H) *unsynchronized* adversary. Further results have been found for other models with weakened synchrony constraints.

Chapter 5

Joint Encryption and Message-Efficient Secure Computation

5.1 Introduction

This chapter connects two areas of recent cryptographic research: secure distributed computation, and group-oriented cryptography. The problem of securely evaluating an arbitrary boolean circuit under cryptographic assumptions has been much studied, beginning with the work of Yao [140] and Goldreich, Micali, and Wigderson [82]. The notion of group-oriented cryptography, in which the power of a secret key holder is distributed over a number of participants, was introduced by Desmedt [58].

Practical implementations of group-oriented public-key encryption were given by Desmedt and Frankel [59]. (See also the related notion of fair public-key encryption [112].) We extend their implementations to achieve additional useful properties. Our scheme, which we call “additive joint encryption,” can then be used to reduce the message complexity of cryptographic multi-party circuit evaluation.

An additive joint encryption scheme enables a group of parties to have individual public keys, such that a message can be encrypted using the keys of any subset of parties. It is easy for any party to “withdraw” from an encryption, and the cooperation of all (current) participants is needed to decrypt. It is easy for each participant to give a “witness” of its contribution to the decryption, so that full decryption can occur in parallel (i.e., anyone can decrypt after seeing all the witnesses). Encrypted messages can be blinded (i.e., replaced by a random encryption of the same message), and they are xor-homomorphic (i.e., from the encryption of two bits it is easy for anyone to compute an encryption of their exclusive-or).

We present an implementation of additive joint encryption with the critical property that the size of an encrypted bit is independent of the number of parties participating in the encryption. This “compact” implementation is a construction that combines El-Gamal public-key encryption and schemes based on quadratic residues and non-residues.

We demonstrate the use of our encryption scheme as a tool for designing efficient multi-party cryptographic protocols (i.e., communication via broadcast channels only). We show that, using

additive joint encryption, a factor of n can be gained in the number of bits broadcast by n parties to securely compute a circuit of size C . Specifically, in the privacy setting (against a passive adversary), only $O(nC)$ encrypted bits of communication are needed, as opposed to $O(n^2C)$ encrypted bits using existing methods.

For specific functions, further gains are possible by exploiting particular connections between properties of additive joint encryption and properties of the functions themselves. We illustrate this for the problem of comparing bit-strings.

Definitions and models are given in Sections 5.2 and 5.3. In Section 5.4, we describe the construction of our new encryption scheme. In Sections 5.5 and 5.6, we demonstrate the application of our scheme to reducing the message complexity of secure multi-party computation. Conclusions and some open problems are given in Section 5.7.

5.2 Model

We assume that there are n parties, each of which is a probabilistic polynomial time Turing Machine (read-only input tape, write-only output tape, random tape, one or more work tapes). The parties communicate by means of a broadcast channel, which can be modeled as an additional tape (communication tape) for each machine that is write-only for its owner and read-only for everyone else. When a party writes a message to this tape, we may say that the message has been “broadcast” or “posted.” A protocol begins with all n parties in their start states, and ends when all have reached their final states. The output of the protocol is the (common) value written on the output tapes of the processors.

We are concerned with the message complexity of a protocol. This is measured as the total number of bits written on the communication tapes during the execution of the protocol. Since our protocols are cryptographic, we will state the message complexity in terms of the number of *encrypted bits* written on the communication tapes. For the protocols we consider, this is all or most of the communication that occurs, and it is also a convenient measure independent of advances in either encryption methods or cryptanalytic techniques.

We will say that a protocol is “private” if its execution reveals no useful information to any subset of (polynomial bounded) gossiping processors. More specifically, anything that is efficiently computable from the views of a subset S of participants is also computable from just the output of the protocol together with the inputs and private keys of S .

5.3 Additive Joint Encryption

In this section we give definitions for additive joint encryption, and then a naive implementation for which the size of an encrypted bit depends on the number of participating parties.

5.3.1 Definition

A *joint encryption scheme* for $[1 \cdots n]$ is a collection of encryption functions $\{E_S : S \subseteq [1 \cdots n]\}$ and a collection of partial decryption functions $\{D_i : i \in [1 \cdots n]\}$ such that $D_i(E_S(M)) = E_{S-\{i\}}(M)$ for all messages M and all $i \in S$, and such that it is easy to compute M from $E_\emptyset(M)$. We use the

notation $D_S(c)$ to stand for the result of applying to c the decryption functions corresponding to each element of S .

A joint encryption scheme is *public-key* if each E_S is easy to compute, while computing each D_i requires a different trapdoor; in fact, our implementation has the stronger “upward conversion” property: $E_{S \cup S'}(M)$ is easy to compute from $E_S(M)$ for any subset S' . A joint public-key encryption scheme is probabilistic if each E_S is probabilistic; we write $E_S(M)$ to denote a uniformly random choice of possible encryptions of M . A probabilistic public-key joint encryption scheme is *secure* if each E_S is GM-secure [85] (computational indistinguishability of ciphertexts, even given the decryption functions $\{D_i : i \in S'\}$ for any $S' \subset S$).

A probabilistic joint encryption scheme is *blindable* if, given any subset $S \subseteq [1 \dots n]$, and an encryption $C = E_S(M)$, it is easy to sample efficiently and uniformly from the set $\{C' : D_S(C') = M\}$ of all possible encryptions of M . We write $blind(C)$ to denote a random choice from this set.

A joint encryption scheme is *xor-homomorphic* if, given any messages $M, M' \in \{0, 1\}^k$, any subset $S \subseteq [1 \dots n]$, and any encryptions $E_S(M), E_S(M')$, it is easy to compute $E_S(M \oplus M')$.

A joint encryption scheme is *witnessed* if there are functions $\{W_i : 1 \leq i \leq n\}$ such that each W_i is hard to compute without the trapdoor for D_i , and such that $D_i(E_S(M))$ can be easily computed from $E_S(M)$ and $W_i(E_S(M))$ for any M and any $i \in S$, but no other $D_i(E_S(M'))$ can be easily computed from $E_S(M')$ and $W_i(E_S(M))$. If every participant in an encryption provides a witness in parallel, the decryption can be computed much faster than by the use of (inherently sequential) partial decryption functions.

Finally, we use the term *additive joint encryption scheme* to denote a secure, blindable, xor-homomorphic, witnessed probabilistic public-key joint encryption scheme.

Desmedt and Frankel [59] consider “threshold” encryption, which is essentially a public-key joint encryption scheme for which any sufficiently large subset of parties can decrypt a message. We do not include this property in our definition, because it is not needed for our main application, secure circuit evaluation. In Section 5.4, we explain how to add threshold decryption capability to our implementation using their techniques.

Notation: We may abuse the xor symbol by extending it to encryptions in the obvious way. When $\hat{x} = E_S(x), \hat{y} = E_S(y)$ are encryptions, we may write $\hat{x} \oplus \hat{y}$ to denote $E_S(x \oplus y)$.

5.3.2 Naive Implementation Based on Xor Shares

A naive implementation of additive joint encryption can be based on any probabilistic public-key encryption scheme that is itself blindable and xor-homomorphic. For example, each encryption function e_i could be an instance of the scheme due to Goldwasser and Micali [85], based on quadratic residues and nonresidues, using a distinct modulus N_i . Let d_i be the decryption function corresponding to e_i .

An additive joint encryption scheme can be constructed as follows. Encryption is given by $E_S(b) = (b', [e_j(b_j) : j \in S])$, where $b_j \in_R \{0, 1\}$ for all $j \in S$, and where $b' = b + \sum_{j \in S} b_j \pmod 2$. Decryption is given by $D_i(\alpha, [\beta_j : j \in S]) = (\alpha \oplus d_i(\beta_i), [\beta_j : j \in S - \{i\}])$ whenever $i \in S$ (or just $d_i(\beta_i)$ can be given as a decryption witness).

Note that the size of an encryption in this scheme grows as the number of participants increases. We seek a compact scheme for which the size of an encryption is independent of the number of parties.

5.4 Compact Additive Joint Encryption

In this section, we show how additive joint encryption can be implemented compactly, i.e., such that the size of an encrypted bit does not depend on the number of parties participating in the encryption.

5.4.1 Intuition of El-Gamal Based Scheme

As shown by Desmedt and Frankel [59], it is possible to construct a joint encryption scheme from El-Gamal's method of public-key encryption [64]. Although blindable, this scheme is not xor-homomorphic, and thus not an additive joint encryption scheme. However, we can convert this into an additive joint encryption scheme for which the size of an encryption is independent of the number of parties.

The basic idea is to use El-Gamal with a composite modulus (whose factorization is unknown), encrypting zeros and ones as El-Gamal encryptions of random quadratic residues and non-residues, respectively (all with Jacobi symbol $+1$). This almost works as is, since blinding and xor can be achieved by component-wise multiplication of the two parts of an El-Gamal encryption. Unfortunately, although these products preserve the correct quadratic character (residue or non-residue) of the encrypted values, the parties—who don't know the factorization of the modulus—will be unable to make use of them. The parties cannot compute the quadratic character of an El-Gamal decryption, unless the entire history of blindings is stored and revealed. It wouldn't help to give the factorization of the modulus to the parties, since that would allow any party to decrypt on its own.

This problem can be solved by accompanying each encrypted bit with an encryption of the witness that allows its decryption. When s^2 is used to encrypt a zero, or $-s^2$ is used to encrypt a one, then the encryption of s^2 or $-s^2$ is accompanied by an El-Gamal encryption of s . This accompanying information makes decrypted values identifiable as residues or non-residues without knowing the factors of N . We give details in the next section.

5.4.2 Details of El-Gamal based Scheme

The public key is $[N, g^{x_1} \bmod N, \dots, g^{x_m} \bmod N]$, where $g \in Z_N^*$, $N = pq$, $p \equiv q \equiv 3 \pmod{4}$, although the prime factors p, q are unknown. The trapdoor information for D_i is x_i . Encryption of a zero is given by

$$E_S(0) = [g^r \bmod N, g^{r'} \bmod N, s^2(\prod_{j \in S} g^{x_j})^r \bmod N, s(\prod_{j \in S} g^{x_j})^{r'} \bmod N]$$

for $r, r', s \in_R Z_N^*$. Encryption of a one is given by

$$E_S(1) = [g^r \bmod N, g^{r'} \bmod N, -s^2(\prod_{j \in S} g^{x_j})^r \bmod N, s(\prod_{j \in S} g^{x_j})^{r'} \bmod N]$$

for $r, r', s \in_R Z_N^*$. Decryption is given by $D_i([\alpha, \beta, \gamma, \delta]) = [\alpha, \beta, \gamma\alpha^{-x_i} \bmod N, \delta\beta^{-x_i} \bmod N]$. Notice that the fourth component of $E_\theta(b)$ enables the quadratic character of the third component (and hence the value of b) to be computed easily.

If $E_S(b) = [\alpha, \beta, \gamma, \delta]$ and $E_S(b') = [\alpha', \beta', \gamma', \delta']$, then

$$E_S(b \oplus b') = [\alpha\alpha' \bmod N, \beta\beta' \bmod N, \gamma\gamma' \bmod N, \delta\delta' \bmod N];$$

thus this scheme is xor-homomorphic. If $[\alpha, \beta, \gamma, \delta]$ is a joint encryption using E_S , and $r, r' \in_R Z_N^*$, then $[\alpha g^r \bmod N, \beta g^{r'} \bmod N, \gamma(\prod_{j \in S} g^{x_j})^r \bmod N, \delta(\prod_{j \in S} g^{x_j})^{r'} \bmod N]$ is a random joint encryption of the same value; thus this scheme is blindable. If $E_S(b) = [\alpha, \beta, \gamma, \delta]$, then $D_i(E_S(b))$ can be easily computed from $[\alpha^{-x_i} \bmod N, \beta^{-x_i} \bmod N]$ for any $i \in S$; thus this scheme is witnessed. The size of each encrypted bit is four elements of Z_N^* , independent of the number of participants; thus this scheme is compact.

We note that our implementation can be modified to include the property of threshold decryption, i.e., encryption such that any sufficiently large subset of parties can decrypt. This can be done, for example, by incorporating the modified shadow generation scheme based on Lagrange interpolation developed by Desmedt and Frankel [59] (which is possible when g and N are chosen as described in the next section).

5.4.3 Security of El-Gamal Based Scheme

Theorem 5.1 *If El-Gamal encryption with a composite modulus is GM-secure, then our compact encryption scheme is GM-secure.*

Proof : Suppose, for purposes of establishing a contradiction, that El-Gamal encryption with a composite modulus is GM-secure while our compact additive joint encryption scheme is not GM-secure. Then it would be easy to distinguish between composite El-Gamal encryptions of $+1$ and -1 , since these can be easily converted into random additive joint encryptions of one and zero, as follows.

Let $(g^r \bmod N, (-1)^b g^{rx} \bmod N)$ be a composite El-Gamal encryption of $(-1)^b$ (using El-Gamal public key $g^x \bmod N$). Then $[g^r \bmod N, g^{r'} \bmod N, s^2(-1)^b g^{rx} \bmod N, sg^{r'x} \bmod N]$ is an additive joint encryption $E_S(b)$ for random $r', s \in_R Z_N^*$ (e.g., using joint public key

$$[N, r_1, \dots, r_{|S|-1}, (g^x \prod_{i < |S|} r_i^{-1} \bmod N)]$$

for random $r_1, \dots, r_{|S|-1}$). \square

However, our encryption scheme (and composite El-Gamal) is not GM-secure if composite quadratic character (residue vs. non-residue) is easy to compute. The attacker sees $g^x \bmod N, \alpha = g^r \bmod N, \beta = g^{r'} \bmod N, \gamma = (-1)^b s^2 g^{rx} \bmod N, \delta = sg^{r'x} \bmod N$, where b is the value of the encrypted bit (and where $x = \sum_{i \in S} x_i$). Let $QR_N(v) = 0$ if v is a quadratic residue modulo N and 1 otherwise. If $QR_N(\cdot)$ is easy to compute, then the attacker can determine $b = (QR_N(\alpha) * QR_N(g^x \bmod N)) \oplus QR_N(\gamma)$. We do not know whether the additional information available to the attacker makes the GM-security of our scheme (and composite El-Gamal) strictly weaker than the difficulty of computing quadratic character modulo N .

The security of the original El-Gamal public-key encryption scheme reduces to the difficulty of breaking an instance of the Diffie-Hellman key exchange scheme [60] (i.e., a problem that is no more difficult than but not known to be equivalent to the discrete log problem). McCurley [109] showed how El-Gamal encryption with a composite modulus (and a careful choice of g and N) can be

secure against an adversary who could break the Diffie-Hellman key exchange, or could factor the modulus, but not both. However this was a proof of security in the sense that no polynomial time algorithm can invert a non-negligible fraction of ciphertexts, and not GM-security (computational indistinguishability of ciphertexts).

We note that if g and N are chosen as suggested by McCurley, then the technical condition for incorporating threshold decryption into our scheme can be met. Specifically, McCurley’s proof is based on the choices $g = 16$, $N = pq$, $p = 8r + 3$, $q = 8s - 1$ (where r, s have special structure), and this meets the condition of Desmedt and Frankel [59] for their modified shadow generation scheme based on Lagrange interpolation (i.e., that g have odd order in Z_N^*).

5.5 Message-Efficient General-Purpose Secure Computation

Several solutions have been found to the problem of securely evaluating an arbitrary boolean circuit under cryptographic assumptions, beginning with the work of Yao [140] and Goldreich, Micali, and Wigderson [82]. We focus on the message complexity (i.e., number of encrypted bits of communication) of such protocols in the privacy setting.

5.5.1 Previous Approaches to Private Computation

Previously, the lowest message complexity known for n parties to privately evaluate a circuit of size C under reasonable cryptographic assumptions was $O(n^2C)$ encrypted bits of communication. This same complexity was achievable using either of the main techniques for secure circuit evaluation in the cryptographic setting: the “gate-by-gate” approach or the “circuit-scrambling” approach.

In the gate-by-gate approach, each gate of the circuit is computed by having each pair of the n parties perform a private two-party protocol. In the protocol of Galil, Haber, and Yung [78], with efficiency improvements by Goldreich and Vainish [83], each two-party protocol is a single instance of “One out of Two Oblivious Transfer” (1-2-OT). It is possible to implement two-party 1-2-OT privately using a constant number of encrypted bits under a cryptographic assumption (e.g., three encrypted bits suffice under the assumption that composite quadratic character is hard). This gives a total message complexity of $O(n^2C)$ encrypted bits.

In the circuit-scrambling approach, each party takes a turn modifying the truth tables of the gates of the circuit. In the protocol of Chaum, Damgård, and van de Graaf [42], each party can randomly permute the rows, and can randomly complement certain of the rows and columns of each truth table. Records of each party’s modifications are preserved in the form of bit commitments, which accompany the scrambled circuit as it passes from party to party (to enable circuit evaluation after the n th party has finished scrambling). Each party contributes a constant number of bit commitments for each gate (e.g., one bit commitment for each truth table row), and so the scrambled circuit as it passes from party i to party $i + 1$ includes $O(iC)$ bit commitments. When each bit commitment is a single encryption, this gives a total message complexity of $O(n^2C)$ encrypted bits.

5.5.2 Reduced “Gate-by-Gate” Message Complexity

A gain of $O(n)$ in the message complexity of secure computation can be achieved via compact additive join encryption using either a gate-by-gate approach or a circuit-scrambling approach. We describe the gate-by-gate approach in detail in this section.

Theorem 5.2 *Under the assumption that compact additive joint encryption is possible, any boolean circuit with C gates can be privately evaluated by n parties using $O(nC)$ encrypted bits of communication.*

Proof : No communication is required for each NOT gate. Each AND gate requires two rounds of communication, and message complexity $4n$ encrypted bits (actually, three encryptions and two decryption witnesses per party, where each witness is half the length of an encryption for the El-Gamal based scheme).

The protocol begins with encryptions of the input bits on the shared tape. We show how the encrypted output of any gate can be computed in a constant number of rounds from its encrypted inputs. For a NOT gate, the output can be found without any communication by XORing the encrypted input with a default encryption of a one.

For an AND gate, suppose the encrypted gate inputs are $E_{[1\dots n]}(x) = \hat{x}$ and $E_{[1\dots n]}(y) = \hat{y}$. Each party i chooses $b_i, c_i \in_R \{0, 1\}$, and broadcasts $\hat{b}_i = E_{[1\dots n]}(b_i)$ and $\hat{c}_i = E_{[1\dots n]}(c_i)$. With no communication, the parties can then find $\hat{x}' = E_{[1\dots n]}(x \oplus b_1 \oplus \dots \oplus b_n)$ and $\hat{y}' = E_{[1\dots n]}(y \oplus c_1 \oplus \dots \oplus c_n)$. The parties broadcast decryption witnesses for \hat{x}', \hat{y}' to find $x' = x + \sum_{1 \leq j \leq n} b_j \pmod 2$ and $y' = y + \sum_{1 \leq j \leq n} c_j \pmod 2$. Let $z' = x' \wedge y'$.

For every $1 \leq i, j \leq n$, party i can find $E_{[1\dots n]}(b_i \wedge c_j)$ by either encrypting a zero (if $b_i = 0$) or by blinding \hat{c}_j (if $b_i = 1$). Similarly, each party i can find $E_{[1\dots n]}(b_i \wedge y)$ and $E_{[1\dots n]}(x \wedge c_i)$. Each party broadcasts a blinded encryption of the XOR of all of these encrypted values (in parallel with the previous broadcast). Now an encryption of the XOR of all received encrypted XOR's, together with an encryption of z' , is equal to the encryption of $z = x \wedge y$ (i.e., by the distributivity of AND over XOR: $u \wedge (v \oplus w) = (u \wedge v) \oplus (u \wedge w)$).

When the last gate in the circuit has been computed, all parties know a joint encryption of the circuit output. At this point, the parties broadcast decryption witnesses to enable all of them to compute the actual circuit output.

For privacy, we need to argue that no subset S of parties learns anything about the other parties' inputs beyond what is implied by a knowledge of the inputs of S and the circuit output. It suffices to show that the distribution of transcripts of protocol executions, as viewed by S , can be simulated by a polynomial time machine that has access to the inputs and decryption functions of S , such that the simulated distribution and the actual distribution are computationally indistinguishable. Excluding the decryption witnesses for the circuit output, the transcript of an execution of the protocol gives a number of joint encryptions for related values, and a number of decryption witnesses for uniformly random values. The simulator computes joint encryptions of uniformly random values to substitute for all of the joint encryptions in the transcript except the last one, and substitutes a uniformly random joint encryption of the output for the joint encryption of the output of the last gate, and substitutes decryption witnesses for uniformly random values for all of the decryption witnesses. By standard cryptographic arguments, the computational indistinguishability of these distributions follows from the computational indistinguishability of individual joint encryptions. \square

5.5.3 Reduced ‘‘Circuit-Scrambling’’ Message Complexity

To get the same gain in message complexity with a circuit-scrambling approach, the bit commitments are additive joint encryptions, but only a single commitment accompanies each truth table row as it passes from party to party. The single commitment at a row represents the XOR of

modifications performed by all parties. When a compact scheme is used, the size of the scrambled circuit doesn't increase from scramble to scramble.

Although the order of magnitude of the message complexity is the same, note that the multiplicative constant is better for our methods using the gate-by-gate approach. In the gate-by-gate approach, four encrypted bits are needed per AND gate (counting one decryption witness as half an encryption), and no communication is needed per NOT gate. In the circuit-scrambling approach, the same four encrypted bits are needed per AND gate, while two encrypted bits are needed per NOT gate (i.e., one bit commitment for each truth table row).

5.5.4 Measures of Message Complexity

In this section, we have shown that the message complexity of secure computation can be decreased by a linear factor in the number of participants. This gain has been computed under a “broadcast” measure of message complexity. Specifically, if one party posts a bit to the publicly readable bulletin board, then the protocol is charged one bit. The same charge applies no matter how many of the other parties ever read that posted bit.

It is reasonable to consider an alternative “readership” measure of message complexity, in which the number of readers of a message is relevant. By this measure, the protocol is charged k bits if a single posted bit is read by k of the other parties.

The linear gain in message complexity is maintained with respect to the readership measure for the “circuit-scrambling” protocol described in Section 5.5.3. This is because most broadcasts are only used to pass the scrambled circuit from one party to the next, i.e., to be read by only one other party. However, the “gate-by-gate” protocol loses the linear gain with respect to the readership measure. Posting messages that are read by all other parties seems to be an essential feature of this approach.

5.6 Customized Secure Protocol for Bit-String Comparison

In this section, we show further application of our encryption method by describing a novel protocol for n parties to privately compare two encrypted bit-strings. The message complexity of this protocol has the same order of magnitude as would be achieved by computing a comparison circuit using our general techniques, although a small constant factor (roughly three) is saved. We believe that this protocol is of interest because it demonstrates that for practical applications (where constant factors count) useful gains in communication complexity can come from customizing cryptographic tools to the specific secure computational task at hand. Note that a constant factor is also gained by this protocol with respect to the readership measure of message complexity.

Before giving our comparison protocol, we need to develop a tool to randomly permute pairs of encrypted inputs with low message complexity.

5.6.1 Shuffle Gate Computation

A “shuffle gate” has two main inputs x, y , a control input c , and two outputs α, β . When $c = 0$, the inputs pass through the gate unchanged: $\alpha = x$ and $\beta = y$. When $c = 1$, the inputs are flipped as they pass through the gate: $\alpha = y$ and $\beta = x$. A shuffle gate can be represented as a circuit with

six AND and OR gates. Using our gate-by-gate approach, $24n$ encrypted bits of communication are needed to privately evaluate a shuffle gate.

By contrast, a uniformly random shuffle gate can be privately computed directly at a cost of only $2n$ encrypted bits of communication using the El Gamal based scheme. Let \hat{x}, \hat{y} be the encryptions of the main inputs. Each party i chooses a uniformly random $c_i \in_R \{0, 1\}$. We want x, y to be flipped only if the xor of all of the c_i values is one. Each party i posts two encrypted values as follows: two encrypted zeros if $c_i = 0$, and two encryptions of $x \oplus y$ if $c_i = 1$. By xoring all of these posted pairs to the input pair \hat{x}, \hat{y} , each party gets an encryption of the appropriate output pair.

5.6.2 Details of the Comparison Protocol

Intuitively, the comparison protocol works on l -bit strings in $l - 1$ rounds, where each round reduces the length of the encrypted bit strings by one. The reduction is done in such a way that the result of comparing the two decrypted bit-strings is preserved after each round. Specifically, the parties remove the leading bits if these bits are equal, and remove the next-to-leading bits if the leading bits are unequal. Of course, it would violate privacy if any proper subset of parties could determine which of these two actions occurred in any round.

Our protocol guarantees that the correct action occurs obliviously (i.e., so that no proper subset of the parties can detect which action occurred) by repeatedly using the shuffle gate construction described in the preceding section. In each round, two shuffle gates are controlled by the same control bit γ . The decision about which pair of bits to discard each round depends critically on the value of the control bit for that round. The details are given in Figure 1.

All messages for this protocol, except for the last round, are witnesses for joint encryptions of uniformly random values, or joint encryptions of related values. Privacy follows from an argument similar to that of Theorem 5.2.

Theorem 5.3 *Under the assumption that compact additive joint encryption is possible, private multi-party comparison of two l -bit strings is possible with communication $4(l - 1)n$ encrypted bits and ln decryption witnesses.*

5.7 Summary and Open Problems

The message complexity of secure distributed computation can be reduced by extending techniques from group-oriented cryptography. We show how gains in multi-party evaluation of general circuits can be achieved by augmenting a joint encryption scheme to support blinding, witnessing, and adding ciphertexts *without* increasing the length of the ciphertexts.

We would like to find compact implementations of additive joint encryption based on other, possibly weaker, intractability assumptions, and to find other applications for such encryption schemes. In addition, we would like to explore other ways to improve the secure evaluation of specific useful functions by exploiting special properties of customized encryption methods. It would also be interesting to reduce the computational resources required for secure computation in other settings, possibly tolerating stronger adversaries.

Initially, $\tilde{x} = E_{[1 \dots n]}(x)$, $\tilde{y} = E_{[1 \dots n]}(y)$.

1. In Round i , $1 \leq i \leq l - 1$, the following messages are sent:

- (a) Each party j ($1 \leq j \leq n$) posts $[\alpha_{1j}, \beta_{1j}, \alpha_{2j}, \beta_{2j}; \gamma_j] =$
 - i. $[blind(\tilde{x}.i \oplus \tilde{x}.(i+1)), blind(\tilde{y}.i \oplus \tilde{y}.(i+1)), blind(\tilde{x}.i \oplus \tilde{x}.(i+1)),$
 $blind(\tilde{y}.i \oplus \tilde{y}.(i+1)); E_{[1 \dots n]}(1)]$ with prob $\frac{1}{2}$.
 - ii. $[E_{[1 \dots n]}(0), E_{[1 \dots n]}(0), E_{[1 \dots n]}(0), E_{[1 \dots n]}(0); E_{[1 \dots n]}(0)]$ with prob $\frac{1}{2}$.
- (b) Each party j ($1 \leq j \leq n$) posts a witness for $D_j(\tilde{x}.i \oplus \tilde{y}.i \oplus \gamma)$, where $\gamma = \gamma_1 \oplus \dots \oplus \gamma_n$.
- (c) Before the start of Round $i + 1$, each party (without communication) locally replaces $\tilde{x}.(i+1), \tilde{y}.(i+1)$ with
 - i. $blind(\alpha_{21} \oplus \dots \oplus \alpha_{2n} \oplus \tilde{x}.(i+1)), blind(\beta_{21} \oplus \dots \oplus \beta_{2n} \oplus \tilde{y}.(i+1))$
if $D_{[1 \dots n]}(\tilde{x} \oplus \tilde{y} \oplus \gamma) = 0$.
 - ii. $blind(\alpha_{11} \oplus \dots \oplus \alpha_{1n} \oplus \tilde{x}.i), blind(\beta_{11} \oplus \dots \oplus \beta_{1n} \oplus \tilde{y}.i)$
if $D_{[1 \dots n]}(\tilde{x} \oplus \tilde{y} \oplus \gamma) = 1$.

2. In Round l , the following messages are sent:

- (a) Each party j ($1 \leq j \leq n$) posts a witness for $D_j(\tilde{x}.l)$.
- (b) Without communication, each party computes the final answer to be $v = D_{[1 \dots n]}(\tilde{x}.l)$. (If $v = 1$ then $x \geq y$ else $x \leq y$.)

Figure 5.1: Message-Efficient Multi-party Comparison Scheme

Chapter 6

Off-Line Electronic Cash

An electronic (or “digital”) coin scheme is a set of cryptographic protocols for withdrawal (by a customer from the bank), purchase (by a customer to a vendor), and deposit (by a vendor to the bank), such that the security needs of all participants are satisfied – money is unforgeable, un reusable, and untraceable. A coin scheme is “off-line” if, as with non-digital metal coins, the purchase protocol does not involve the bank. No off-line electronic coin scheme has yet been proposed which is both provably secure with respect to natural cryptographic assumptions and efficient with respect to reasonable measures.

We prove that two primitives are sufficient to implement a secure off-line electronic coin scheme: Embedding scheme and Oblivious Authentication. Previous coin schemes have used techniques that can be viewed as heuristic implementations of these primitives.

We present an Embedding scheme whose security depends only on the hardness of the discrete log function. Combining our Embedding scheme with a heuristic Oblivious Authentication scheme yields a new approach to efficient (but not provably secure) off-line coins.

We give a protocol for oblivious authentication that has a pre-processing stage (preparatory, independent of the transactions) and an efficient main stage (at actual withdrawal time). The pre-processing stage can be implemented based on general cryptographic assumptions with high degree polynomial communication. Alternatively, pre-processing can be implemented efficiently assuming the presence of an initiating trusted agent. The only role of this agent is to produce generic strings (“blank withdrawal slips”) that are sent to customers. Many cryptographic schemes (e.g., identification schemes) use initiating servers in a similar role; as with these other schemes, the availability of an initiator does not trivialize the tricky security and correctness requirements of the underlying problem.

Combining our Embedding scheme and our Oblivious Authentication protocol yields an off-line coin scheme that is efficient in the sense that we put forth in this work: “a protocol is efficient if its communication complexity is independent of the computational complexity of its participants” (and thus the communication length and number of encryption operations is only a low-degree polynomial of the input). We are motivated by the fact that, typically, such protocols are easily modifiable to produce a working practical variant (based perhaps on specialized assumptions and somewhat related heuristic tools, while keeping the basic protocol structure and computations).

6.1 Introduction

6.1.1 What is an Off-Line Electronic Coin Scheme?

An electronic coin scheme [39] is a set of cryptographic protocols for withdrawal (by a customer from the bank), purchase (by a customer to a vendor while possibly involving the bank), and deposit (by a vendor to the bank), such that the security needs of all participants are satisfied: anonymity of purchase for the customer, assurance of authenticity for the vendor, impossibility of undetected reuse or forgery for the bank.

A coin scheme has the interesting “off-line” property [43] if the purchase protocol does not involve the bank; everyday non-digital money is of course off-line. To balance the customer’s need for anonymity (so that her/his spendings, and thus her/his lifestyle, cannot be traced by the bank/government) with the bank’s need for protection against reuse (to prevent counterfeit money), each coin in an off-line scheme must somehow “embed” the customer’s identity in a way that is accessible only if the same coin is used for more than one purchase. Moreover, to balance the customer’s need for anonymity with the bank’s need for protection against forgery, each coin must somehow get an authentication from the bank (e.g., a digital signature) without being seen by the bank!

Meeting all of these security needs simultaneously is a difficult task. In fact, no scheme has yet been proposed which is both provably secure with respect to natural cryptographic assumptions and efficient with respect to reasonable measures. We review previous approaches to the problem and present our new approach and contributions.

6.1.2 Previous Approaches to Off-Line Coin Schemes

There have been two main approaches to off-line coin schemes in the literature. One direction is based on blind signatures, and the other is based on zero knowledge proofs of knowledge.

Schemes Based on Blind Signatures

A blind signature scheme of Chaum [39] is a protocol that enables one party (i.e., a customer) to receive a signature of a message of its choice under the second party’s (i.e., the bank’s) private signature key. The second party learns nothing about the message it helped to sign. For use in an off-line coin scheme, the bank must be sure that the message it never sees has a certain form (i.e., that it embeds the customer’s identity in the proper way). This can be done by combining the blind signature scheme with a zero knowledge proof [86, 81] or “cut-and-choose” check [125] that the message has the right form. The first off-line coin scheme, due to Chaum, Fiat, and Naor [43] takes this approach, using the blind signature scheme based on RSA [129]. This approach is efficient but heuristic; no proof of security has been given that relies on assumptions about RSA that are simple or natural. In fact, refinements to their protocol [3] were later found to introduce security flaws [90], which underscores the risk of relying on heuristics. Other off-line coin schemes use blind signature schemes derived from Chaum and Pedersen [44].

It is also possible to implement blind signatures using general secure 2-party computation protocols [140], as shown by Pfitzmann and Waidner [124], building on work of Damgård [56]. A circuit to compute signatures is jointly computed by the bank and the customer, with the bank contributing one input (secret signature key), the customer contributing the other input (message

to be signed), and the customer alone receiving the output (signed message). The security of a scheme of this type can be reduced to general cryptographic assumptions. However, the message complexity (number of bits sent between parties) and encryption complexity (number of applications of encryption operations) of all known secure computation protocols is proportional to the size of the circuit being computed. It is unreasonable for a coin scheme to have a communication cost that depends on the computational complexity of the underlying signature function algorithm (which should be only a “local” computation for the bank)—the same way it is unreasonable to have a one cent coin weigh fifty kilograms. Thus, a scheme of this type is secure, but definitely not efficient.

Schemes Based on Zero Knowledge Proofs of Knowledge

A zero knowledge proof of knowledge [86] is a protocol between a prover and a verifier, in which the verifier is convinced that the prover possesses a witness (e.g., for membership in a language) without learning anything about that witness. In a non-interactive zero knowledge proof of knowledge [57], a single message is sent (from prover to verifier); in this case, the two parties are assumed to share a random string. DeSantis and Persiano [57] show that, when non-interactive zero knowledge proofs of knowledge are possible, blind signatures are not necessary for an off-line coin scheme (see also [117]). The basic idea is that the bank gives an ordinary signature to the customer, but no one ever sees that signature again. Instead of presenting the signature to validate a coin at purchase time, the customer presents the vendor with a proof that it possesses a valid signature from the bank. To deposit the coin, the vendor presents the bank with a proof that it possesses a valid proof of possession from the customer. The security of this type of scheme can be proven with respect to general cryptographic assumptions. However, the message complexity and encryption complexity is prohibitively large, e.g, the length of a proof of possession of a valid signature depends on the complexity of the signing function itself. Thus a scheme of this type is secure, but again it is not efficient.

6.1.3 Security and Efficiency: A Closer Look

We want an off-line coin scheme that is both secure and efficient. Let us clarify more formally what we mean by security and by efficiency.

Security of protocols means that the claimed properties can be proven based on some intractability assumption. This is the typical notion of complexity-theoretic security.

Efficiency of protocols, which is a notion we try to capture in this work, can be formalized by requiring that “the communication complexity of the protocol is independent of the computational complexity of the honest participants”. In schemes which lead to practical implementations, like identification schemes [69] [72] and basic practical operations like signature and encryption, this is the case (the communication is a low-degree polynomial function of the security parameter and the input size). On the other hand, general protocols which encrypt computations of results (like general zero-knowledge schemes for NP and IP, and secure computation schemes) are considered impractical by users of cryptography, since the local computation is typically a much higher-degree polynomial function of the input and the security parameters. This makes them prohibitively expensive – as they require much communication and, more crucially, similarly many applications of cryptographic operations. We feel that it is important to make this distinction to further develop

a sound theory of cryptography with robust notions of efficiency; our effort here is to be considered as a first step in this direction. (We note that there are further measures of efficiency; in particular we also measure and try to minimize the number of rounds).

In this paper, we show that a secure and efficient off-line coin scheme is possible, based on what we call “oblivious authentication” and the hardness of the discrete log function. We implement oblivious authentication (to be described later) based on any one-way function together with a pre-processing stage independent from the withdrawal, purchase, and deposit protocols. The communication complexity of our scheme is $O(k^2s)$ bits where k is a security parameter and s is the size of a signed bit; $O(k^2)$ encryption operations are performed. Purchase and deposit are non-interactive, while withdrawal requires two rounds of interaction.

6.1.4 Off-Line Coin = Passport + Witness + Hints

Next we informally (and metaphorically) describe the building blocks and mechanisms which underlie the efficient off-line digital coin scheme. In real life, a “passport” is an authorized document from a trusted source (the government) that identifies its owner. We can also imagine a passport without any name or picture, but still stamped by the proper authority, allowing a sort of anonymous authentication. Off-line coin schemes make use of yet another type of passport, in which the identity of its owner is only *somewhat* present, embedded in the document in a complexity theoretic sense (through the use of what we call an Embedding scheme).

A primitive that we call Oblivious Authentication lets an authorizing agency issue a digital passport with an embedded secret, together with a “witness” of the embedded secret. The issuing process is oblivious in the sense that the authorizing agency cannot later connect any passport to the time it was issued. Use of the passport without the witness would allow repeated authentication while concealing forever the secret (against a resource bounded attack). Use of the passport with the witness would authenticate the user and reveal the user’s embedded secret. We will use the passport with a “hint” of the witness, where any two hints, but no single hint, is enough to recover the witness¹. When the embedded secret is the identity of the user, this is the crux of an off-line coin scheme (see Figure 6.1).

Withdrawal is an instance of Oblivious Authentication (using an Embedding scheme), in which the bank issues a passport (with embedded identity) and a witness to the customer. To make a purchase, the customer gives the passport and a unique hint (extracted from the witness) to the vendor. To deposit, the vendor forwards the passport and the hint to the bank.

In Section 6.5, we prove that a secure off-line coin scheme can be constructed from any embedding scheme together with any Oblivious Authentication scheme. In particular, our new Oblivious Authentication scheme together with our new embedding scheme yields a provably secure and efficient off-line coin scheme. Other coin schemes are possible through various combinations of our new protocols with previous methods.

6.1.5 Embedding Scheme

The goal of an Embedding scheme is to hide a secret so that it is concealed unless a witness is known, and to allow for the extraction of verifiable hints (any two of which reveal a witness). The

¹We can generalize to hints that reveal the witness at thresholds greater than two, which is useful (e.g., for multiple access authorization tokens) but not considered in this paper.

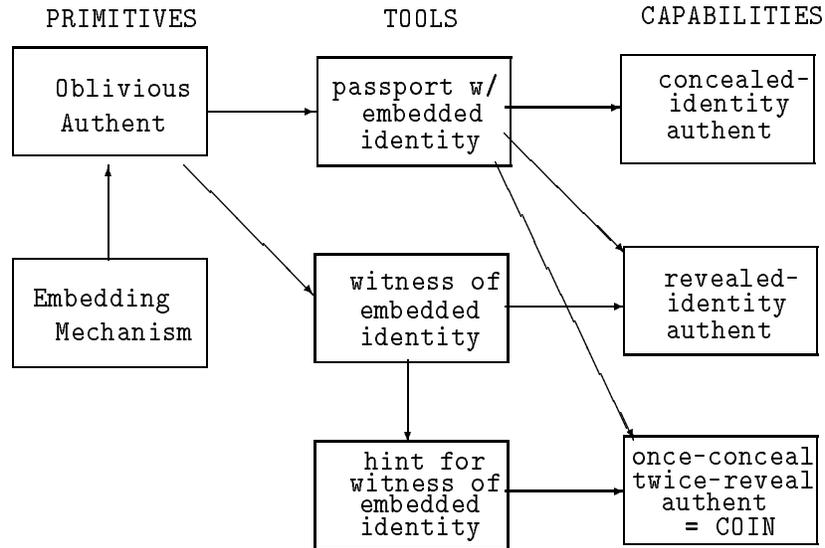


Figure 6.1: Building an off-line coin scheme from basic primitives

only prior Embedding scheme for an efficient off-line coin scheme, due to Chaum, Fiat, and Naor [43], hides the secret in a collection of nested one-way functions that each hide a pair of xor shares of the secret. A hint of the witness is a particular “de-nesting” of the functions in a way that reveals one xor share from each pair (“hint” of the witness). Any two distinct hints will include both xor shares from at least one pair, from which the secret can be recovered. The security of this scheme has never been reduced to natural cryptographic assumptions.

In our method, a secret is hidden as bits of the discrete logs of a number of values. A hint of the witness is a set of points on lines whose slopes are these discrete logs. From any two distinct hints, the secret can be recovered by interpolation of the corresponding pairs of points. The security of this scheme is based on the hardness of the Discrete Log.

6.1.6 Implementing Oblivious Authentication

We implement Oblivious Authentication based on any one-way function together with a pre-processing stage. The purpose of the pre-processing stage is to produce generic strings that are sent to any customer that asks for them. Each string enables a customer to withdraw a single coin from the bank. These strings are generic in the sense that any string could go to anyone, so long as no string goes to more than one party (like “blank withdrawal slips”).

The pre-processing stage of our Oblivious Authentication protocol can be based on the presence of some trusted agent that is separate from the bank, vendors, and customers and is present at initiation. This agent, which we call a “trusted manufacturer” might fill each user’s smart card memory once with a large number of strings, and then destroy its records. Unlike some physically based schemes [67], the smart card memory does not need to be shielded in any way from the

owner of the smart card (i.e., no read or write restrictions are needed). The only issue is that the manufacturer is trusted to produce strings of the proper form, and to never give the same string to more than one party. This mild assumption, as argued above, still leaves the major problems of off-line money scheme security with no trivial answer.

How reasonable is this assumption? Many cryptographic systems need a trusted center. Any complete public key system needs a trusted center of our kind to certify the connection between keys and users without risking impersonation attacks. The zero-knowledge identification scheme of Feige, Fiat, and Shamir [69] uses a trusted center to publish a modulus with unknown factorization and to maintain a public key directory; in the “keyless” version of their system, the trusted center issues tamper-resistant (unreadable, unwritable) smart cards to all participants.

Alternatively, pre-processing can be done inefficiently (high degree polynomial communication), under general cryptographic assumptions, as a general secure protocol between bank and customer. The efficiency of the withdrawal, purchase and deposit protocols would be unaffected. Our stated principle of efficiency would be violated only at initiation, which can be a background computation independent of money-exchanging transactions. When Oblivious Authentication is realized in this way with an inefficient preparatory stage and an efficient main stage, it has a form similar to the “on-line/off-line” signature scheme of Even, Goldreich, and Micali [66].

6.1.7 Structure of the Chapter

Preliminary definitions and notions are given in Section 6.2. Oblivious Authentication is covered in Section 6.3, and Embedding schemes in Section 6.4. Section 6.5 covers off-line coins in a way that relates the notion to the previous two primitives. Implications are discussed in Section 6.6.

6.2 Preliminaries

A quantity is “negligible” in k if it is smaller than $1/p(k)$ for every polynomial p , for k large enough; otherwise, it is “non-negligible.” We let $x \in_R S$ denote the uniformly random selection of x from S . We let $x.i$ denote the i th component of x .

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 (a) there is a deterministic polynomial-time (in k) algorithm to compute each f_k ; and (b) for every probabilistic polynomial time (p.p.t.) algorithm A , the probability that $f_k(A(f_k(x))) = f_k(x)$ is negligible (in k), where the probability is over the uniformly random choice of $x \in \{0, 1\}^k$ and over the random bits used by A .

A “hard bit” for a family of functions $\{f_k\}$ is a family of functions $h_k : \{0, 1\}^k \rightarrow \{0, 1\}$ such that determining $h_k(x)$ from $f_k(x)$ is as hard as determining x from $f_k(x)$ (or negligibly close, in k). Goldreich and Levin [80] showed that every one-way function has a hard bit.

The “Discrete Log Assumption” (DLA) is that exponentiation modulo a prime is a one-way function (i.e., where $f_k(x) = g^x \bmod p_k$ for some prime p_k of length k and some generator g of $Z_{p_k}^*$), or equivalently that it has a hard bit. The difficulty of the discrete logarithm problem is a standard cryptographic assumption first used by Diffie and Hellman [60] (on other basic uses and hard bits of the discrete log, see [30, 106]).

A “communicating p.p.t. Turing Machine” is like a normal p.p.t. Turing Machine (with a read-only input tape, write-only output tape, read-only random tape, one or more work tapes), except

that for each pair of machines there is also a read-only shared input tape, and two communication tapes, each of which is read-only for one machine and write-only for the other.

A “protocol” is the joint execution of a pair of communicating p.p.t. Turing Machines. The “view” of a machine is the contents of all of its private tapes and shared tapes throughout the execution of a protocol.

6.3 Oblivious Authentication

Informally, an oblivious authentication scheme is a protocol between a requestor and an authenticator. The requestor and authenticator begin with the shared secret s , and at the end of the protocol the requestor is able to compute a random (witness) string r and an authenticating (signature) string σ . The relation among these values is that σ is a signature of z in the authenticator’s private key, where $z = e(s, r)$ for some public and easily computable function e (whose purpose will become clear in Section 6.4 on Embedding schemes). The signature reflects authentication that the shared secret s is actually related to the signed value in this way. At the same time, the protocol is “oblivious” from the point of view of the authenticator i.e., after several executions of the protocol, the authenticator must not be able to connect any $[z, \sigma]$ pair to the instance of the protocol that produced it. Using terminology from the Introduction, the authenticator issues a “passport” $[z, \sigma]$ with r serving as a witness to the fact that the secret s is embedded in the passport.

6.3.1 Formal Definition of Oblivious Authentication

Definition 6.1 *An oblivious authentication scheme $[R, A, e]$ is a protocol between a p.p.t. requestor R and a p.p.t. authenticator A , using a p.p.t. computable function e . R and A begin with shared string s , A has a secret key for a signature scheme secure against a chosen message attack, and both A and R have the corresponding public key. R ends up with r, σ , such that the following requirements are satisfied:*

1. (Valid) σ is a valid signature of $z = e(s, r)$ with respect to the Authenticator’s public key.
2. (Oblivious) The protocol leaks no information to A about z or r except that $z = e(s, r)$, i.e.: Let M be any p.p.t. TM that takes as input shared strings s_0, s_1 , random tapes rnd_0, rnd_1 , output information z, σ , views v_0, v_1 of A of executions of the O.A. protocol, and auxiliary input aux , and outputs the guess $b \in \{0, 1\}$. Then there exists a p.p.t. TM M' that takes as input only s_0, s_1, z, σ, aux such that for all s_0, s_1, rnd_0, rnd_1 , if v_0, v_1 are consistent with s_0, rnd_0 and s_1, rnd_1 respectively, and if z, σ is consistent with $v_b \in \{v_0, v_1\}$, then the probability that M outputs a correct guess for b is negligibly close to the probability that M' outputs a correct guess for b .
3. (Unforgeable) From n transcripts of a cheating Requestor, beginning with shared strings s_1, \dots, s_n , it is hard to compute z, σ such that σ is a valid signature of $z = e(s, r)$ under the Authenticator’s secret key for some $s \notin \{s_1, \dots, s_n\}$.
4. (Unexpandable) From n transcripts of a cheating Requestor, it is hard to compute $z_1, \sigma_1, \dots, z_{n+1}, \sigma_{n+1}$ such that each σ_i is a valid signature of $z_i = e(s_i, r_i)$ under the Authenticator’s secret key while each $[r_i, s_i]$ pair is distinct.

1. In Round One, the following messages are sent:
 - (a) $R \rightarrow A: \hat{z}_1 = \rho_1^{e_A} h(z_1) \bmod N_A, \dots, \hat{z}_{2k} = \rho_{2k}^{e_A} h(z_{2k}) \bmod N_A$, where
 - i. $\rho_i \in_R Z_{N_A}^*$ for all $1 \leq i \leq 2k$.
 - ii. $z_i = e'(s, r_i)$ for all $1 \leq i \leq 2k$, where each r_i is uniformly random over the appropriate range, and where e' is a public and easily computable function.
 - iii. h is a publicly known collision-free hash function.
 - (b) $A \rightarrow R: S \subset_R [1 \dots 2k], |S| = k$.
2. In Round Two, the following messages are sent:
 - (a) $R \rightarrow A: [r_i, \rho_i : i \in S]$.
 - (b) $A \rightarrow R: \hat{\sigma} = \prod_{j \notin S} (\hat{z}_j)^{d_A} \bmod N_A$, assuming that the messages received so far are consistent (otherwise A terminates the protocol); i.e.,
 - i. $\hat{z}_j \rho_j^{-e_A} = h(e'(s, r_j))$ for all $j \in S$.
3. R finds $[r, \sigma]$ where
 - (a) $r = [r_j : j \notin S]$
 - (b) $\sigma = \hat{\sigma} (\prod_{j \notin S} \rho_j^{-1}) \bmod N_A$
 - (c) The public and easily computable function e is defined to be $e(s, r) = [e'(s, r[1]), \dots, e'(s, r[k])]$.

Figure 6.2: Heuristic Oblivious Authentication via RSA

6.3.2 Heuristic Implementation of Oblivious Authentication

Oblivious Authentication can be implemented heuristically using multiple blind RSA signatures, collision-free hash functions, and a cut-and-choose challenge procedure (see Figure 6.2). A collision-free hash function is a function h for which it is computationally infeasible to find x, y such that $h(x) = h(y)$.

6.3.3 Provably Secure Oblivious Authentication

In this section, we describe an implementation of oblivious authentication that depends on a pre-processing stage. This pre-processing stage can be done directly by a “trusted manufacturer.” A “trusted manufacturer” is an entity that can perform computations, and can send information to a requestor without it being seen by the authenticator, and without the requestor having to identify itself. For example, data may be put by the manufacturer into the memory of a smart card that is then sold to a requestor. It is not necessary to shield the smart card memory from the requestor, only that the authenticator learns nothing about the data (e.g., by eavesdropping on the transfer of data to the requestor, or by corrupting the manufacturer).

Alternatively, pre-processing can be implemented inefficiently using a general secure 2-party

computation protocol between the requestor and the authenticator. As shown by Kilian [97], the assumption that Oblivious Transfer [126] (a basic cryptographic primitive) is possible suffices.

Intuitively, the pre-processing stage supplies encrypted windows, where each window presigns both a zero and a one for a given bit position in the message. Each signature is concealed in a half-window as a pair of elements, where the first element is a (secret-key) encryption of a random mask, and the second element is the xor of the mask with the signature. In this way, the authenticator is needed to reveal either of the signatures in a window (by decrypting the mask), but the authenticator cannot later recognize a signature he helped reveal.

A collection of m windows (called a “row”) can be used to sign a string of m bits. However, to prevent signed bits from one message being substituted in a second message, the windows must be “linked” together. In fact, all pairs of adjacent half-windows are connected, so that the same collection of windows is capable of signing any possible message. Links are also pairs of elements, where the first element is a (secret-key) encryption of a random mask, and the second element is the xor of the mask with a signature of the two half-windows that are being linked. When half-windows are “opened” (i.e., when their masks are decrypted), the corresponding links are opened as well.

A cut-and-choose procedure is used to guarantee that the predicate $\psi(z, id)$ is true at the end. There will be $2k$ strings that are presented to the authenticator, each of which satisfies some predicate $\psi'(z', id)$, and for each of which the requestor knows a witness wit' . The authenticator challenges half of the strings, and receives witnesses that the predicate is true for them. The authenticator then sends decrypted masks for all half-windows and links for all unchallenged strings. The predicate ψ is defined to hold if and only if a majority of the k unchallenged strings satisfy ψ' .

A second type of link is needed to prevent cheating with the cut-and-choose procedure. Rows of windows need to be linked together, to prevent a new collection of k rows from being pieced together from two or more runs of the protocol. This is achieved by providing links between half-windows at the end of one row to half-windows at the beginning of the next row.

The entire data structure, windows and links, is called a “struct.” We assume that the pre-processing stage (via trusted manufacturer or secure 2-party computation) has sent a supply of structs to the requestor. A pictorial representation of the data structure appears in Figure 6.3, while precise definitions appear in the proof of Theorem 6.1.

The proof of Theorem 6.1 consists of showing that a secure Oblivious Authentication can be based on the struct data structure. Structs are supplied by a pre-processing stage. Since structs contain encrypted bits and signed bits with respect to the Authenticator’s keys, we must assume that these keys are available to the original “manufacturer” of the struct. If the structs are computed as a secure computation with the Authenticator itself, then the keys are automatically available for this computation. If some trusted manufacturer produces the structs, then we are tacitly assuming that this manufacturer has access to the Authenticator’s keys. In this case, a dishonest manufacturer could forge authentications at will.

Theorem 6.1 *If one way functions exist, and a pre-processing stage is allowed, then there exists a secure Oblivious Authentication scheme.*

Proof: First we describe the “struct” data structure that will be used for each instance of the O.A. protocol. Let sig be a signature scheme that is existentially unforgeable against a chosen message attack; such a scheme exists if one way functions exist [116] [130]. Let E be a symmetric-key encryption function, which also exists if one way functions exist [92, 93, 88].

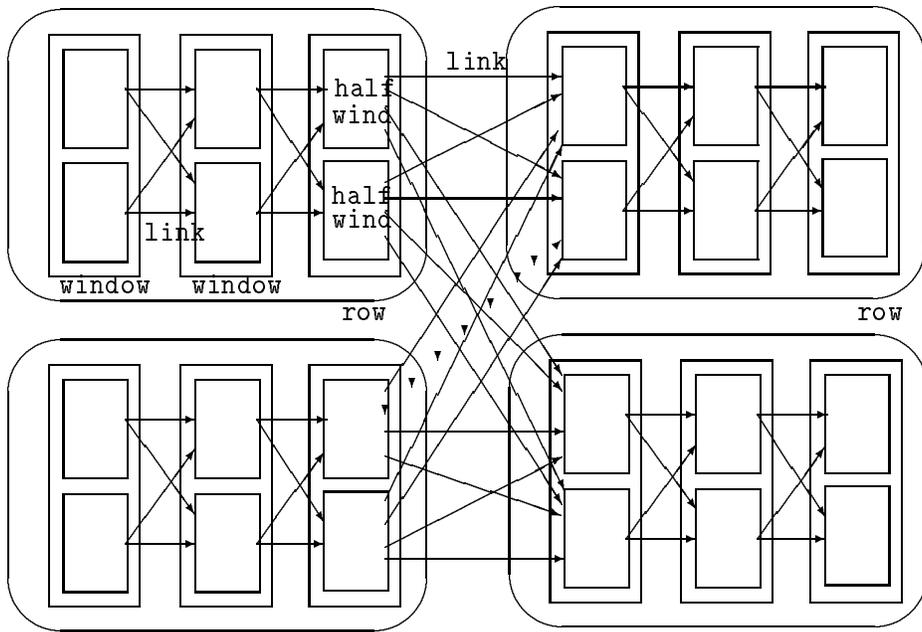


Figure 6.3: Struct Data Structure (informal picture: 4 out of $2k$ rows, $m = 3$)

A **half-window** w_{ijl}^β with serial number ser is of the form $[E(ser||i, j, l, b_{ijl}^\beta||\rho_{ijl}^\beta), b_{ijl}^\beta, \rho_{ijl}^\beta \oplus sig(i, j, \beta||x_{ijl}^\beta)]$. Here the “half-window label” β is a bit, the “half-window pseudolabel” b_{ijl}^β is a random bit, the “half-window value” x_{ijl}^β is a random string, and the “half-window value mask” ρ_{ijl}^β is a random string. We refer to the components of a half-window as the left side (encrypted pseudolabel and mask), pseudolabel, and right side (masked signature) of the half-window.

A **half-window link** $L(w_{ijl}^\beta, w_{i'j'l'}^{\beta'})$ between half-windows $w_{ijl}^\beta, w_{i'j'l'}^{\beta'}$ with serial number ser is of the form $L(w_{ijl}^\beta, w_{i'j'l'}^{\beta'}) = [E(ser||i, j, l, i', j', l' || \rho_{ijl, i'j'l'}^{\beta\beta'}), \rho_{ijl, i'j'l'}^{\beta\beta'} \oplus sig(i, j, x_{ijl}^\beta || i', j', x_{i'j'l'}^{\beta'})]$. Here the “half-window link mask” $\rho_{ijl, i'j'l'}^{\beta\beta'}$ is a random string, and $x_{ijl}^\beta, x_{i'j'l'}^{\beta'}$ are the half-window values of $w_{ijl}^\beta, w_{i'j'l'}^{\beta'}$ respectively. The two components of a half-window link are referred to as the left side (encrypted pseudolabels and mask) and right side (masked signature) of the half-window link.

A **window** w_{ijl} with serial number ser is of the form $w_{ijl} = [w_{ijl}^0, w_{ijl}^1]$, where both half-windows have serial number ser , and where the pseudolabels b_{ijl}^0, b_{ijl}^1 are unequal random bits.

A **row** W_{il} of length m with serial number ser is of the form $W_{il} = [w_{i1l}, \dots, w_{iml}, \{L(w_{ijl}^\beta, w_{i,j+1,l}^{\beta'}) | \beta \in \{0, 1\}, \beta' \in \{0, 1\}, 1 \leq j \leq m-1\}]$, where all windows and links have serial number ser .

A **struct** W of length m with serial number ser and security parameter k is of the form $W = [W_{10}, W_{11}, W_{20}, W_{21}, \dots, W_{k0}, W_{k1}, \{L(w_{iml}^\beta, w_{i+1,1,l}^{\beta'}) | \beta, \beta', l, l' \in \{0, 1\}, 1 \leq i \leq k\}]$, where all rows and links have serial number ser .

The Oblivious Authentication protocol $[R, A, e]$, with shared string s , proceeds as follows:

1. In Round One, the following messages are sent:

- (a) $R \rightarrow A$: the left side of every half-window and half-window link in W , and $[b_{ijl}^\beta | 1 \leq i \leq k, 1 \leq j \leq m, l \in \{0, 1\}, \beta = z_{il}.j]$, where
 - i. W is an unused struct of length m owned by R .
 - ii. R chooses $r_{10}, r_{11}, \dots, r_{k0}, r_{k1}$ uniformly at random, and $z_{il} = e(s, r_{il})$.
- (b) $A \rightarrow R$: $[chall_1, \dots, chall_k] \in \{0, 1\}^k$, assuming that the messages received so far are consistent and correct (otherwise, A terminates the protocol); i.e.,
 - i. The decryptions of the left sides of half-windows and half-window links sent by R are syntactically correct.
 - ii. The decryptions of the left sides of half-windows and half-window links begin with a common serial number ser .
 - iii. The common serial number ser has not been seen by A in any previous execution of the protocol.

2. In Round Two, the following messages are sent:

- (a) $R \rightarrow A$: the right side of every half-window in W_{i, chall_i} for all $1 \leq i \leq k$; the right side of every half-window link $L(w_{i,j, \text{chall}_i}^\beta, w_{i',j+1, \text{chall}_{i'}}^{\beta'})$ for all $1 \leq i, i' \leq k, 1 \leq j < m$; and r_{i, chall_i} for all $1 \leq i \leq k$.
- (b) $A \rightarrow R$: $\rho_{i,j,1-\text{chall}_i}^\beta$ for every $b_{i,j,1-\text{chall}_i}^\beta$; $\rho_{i,j,1-\text{chall}_i, i, j+1, 1-\text{chall}_i}^{\beta\beta'}$ for every $b_{i,j,1-\text{chall}_i}^\beta, b_{i,j+1,1-\text{chall}_i}^{\beta'}$; and $\rho_{i,m,1-\text{chall}_i, i+1, 1, 1-\text{chall}_{i+1}}^{\beta\beta'}$ for every $b_{i,m,1-\text{chall}_i}^\beta, b_{i+1,1,1-\text{chall}_{i+1}}^{\beta'}$ received by A in Round One, assuming that the messages received so far are consistent and correct (otherwise, A terminates the protocol); i.e.,
 - i. For every half-window in every challenged row, the xor of the right side with the mask from the left side yields a valid signature.
 - ii. For every fully revealed half-window link, the xor of the right side with the mask from the left side yields a valid signature, and the half-windows it connects have consistent half-window values.
 - iii. Every z_{i, chall_i} (computable since every fully revealed half-window exposes the correspondence between the pseudolabel b_{ijl}^β and label β) satisfies $z_{i, \text{chall}_i} = e(s, r_{i, \text{chall}_i})$.

3. R finds $[r, \sigma]$ where

- (a) $r = [r_{1,1-\text{chall}_1}, \dots, r_{k,1-\text{chall}_k}]$.
- (b) $\sigma = [\text{sigbits}(z), \text{inlinks}(z), \text{outlinks}(z)]$ where
 - i. $\text{sigbits}(z) = [\text{sig}(i, j, \beta | x_{i,j,1-\text{chall}_i}^\beta) : 1 \leq i \leq k, 1 \leq j \leq m, \beta = z_{i,1-\text{chall}_i} \cdot j]$.
 - ii. $\text{inlinks}(z) = [\text{sig}(i, j, x_{i,j,1-\text{chall}_i}^\beta || i, j + 1, x_{i,j+1,1-\text{chall}_i}^{\beta'}) : 1 \leq i \leq k, 1 \leq j \leq m - 1, \beta = z_{i,1-\text{chall}_i} \cdot j, \beta' = z_{i,1-\text{chall}_i} \cdot j + 1]$.
 - iii. $\text{outlinks}(z) = [\text{sig}(i, m, x_{i,m,1-\text{chall}_i}^\beta || i + 1, 1, x_{i+1,1,1-\text{chall}_{i+1}}^{\beta'}) : 1 \leq i \leq k - 1, \beta = z_{i,1-\text{chall}_i} \cdot m, \beta' = z_{i+1,1-\text{chall}_{i+1}} \cdot 1]$.

4. R accepts if the following Validity Properties are satisfied:

- (a) $\text{sigbits}(z)$ are good signatures with proper indices.
- (b) $\text{Inlinks}(z)$ and $\text{outlinks}(z)$ are good signatures with proper indices and half-window values that are consistent with the half-window values of $\text{sigbits}(z)$.

Given the pre-processing stage (trusted manufacturer, or secure 2-party protocol using Oblivious Transfer), we can assume that the requestor has been given a sufficient supply of valid structs with unique serial numbers. We claim that the protocol described above satisfies the four requirements of an oblivious authentication scheme.

The first requirement (validity) is satisfied with respect to the validity properties checked by R in Step 4 of the protocol.

For the second requirement (obliviousness), it suffices to observe that the view of A of an execution of the protocol can be sampled efficiently by A given only the shared string and its random tape, assuming that A can compute sig and E on its own. The random tape of A only determines the k challenge bits, but these are not relevant to any computation since different challenges could lead to the same r, σ if the half-windows of the struct had been flipped appropriately initially. Thus

the view of A can be sampled with respect to a uniformly random set of challenges without affecting the success of any guessing TM M' .

To address the other requirements, note that the only signatures σ that R can produce to satisfy the validity properties (listed in Step 4 of the protocol) are those that are paths through structs received from the manufacturer. At least one new inlink or outlink would be required to create a new signature σ from old ones, and this is impossible for the requestor to do since the underlying signature scheme sig is existentially unforgeable against a chosen message attack (in fact, it suffices that the signature scheme be existentially unforgeable under a random message attack).

This observation immediately implies that the fourth requirement (unexpandability) is satisfied. This is because the authenticator only un.masks a single path through each of n structs no matter how the Requestor behaves. To create $n + 1$ paths from these would require reuse of some windows, which would require creation of new half-window links that didn't appear in the original unmasked paths. This would violate the existential unforgeability of the underlying signature scheme.

The observation also implies that the third requirement (unforgeability) is satisfied. Since no cheating Requestor can produce a path through a struct other than the n that are unmasked by the Authenticator, the only way to achieve a forgery is with one of those n unmasked paths itself. The Authenticator never un.masks a path unless $z = e(s, r_{i,j})$ for all challenged rows. With probability at least $1 - 2^{-k/2}$, this relation also holds for a majority of the k unchallenged (and subsequently unmasked) entries in the path. With probability at least $1 - n2^{-k/2}$, a majority of the k entries in each of the n unmasked paths embed the appropriate secret. So no matter how the Requestor behaves, with overwhelming probability he is either caught cheating or he ends up with signed elements which embed the appropriate secrets unambiguously. \square

6.4 Embedding Scheme

Informally, an Embedding scheme allows a secret to be hidden so that it can be revealed gradually. Witnesses to the secret can be constructed that are easily verifiable by anyone, such that the secret is revealed by any two witnesses, but inaccessible given only one witness.

6.4.1 Formal Definition of Embedding Scheme

Definition 6.2 *An embedding scheme is a pair of functions e, W such that the following requirements are satisfied:*

1. *(Pseudo-Injective) From s, r, i it is hard to compute any $W(s', r', i)$ such that $e(s, r) = e(s', r')$ while $[s, r] \neq [s', r']$.*
2. *(Once-Concealed) If $z = e(s, r)$, and $h_i = W(s, r, i)$, then for every s, s', i it is hard given s, s', z, i, h_i to distinguish the correct s from the incorrect s' non-negligibly better than random guessing, for all but a negligible fraction of r .*
3. *(Twice-Revealed) If $z = e(s, r)$, $h_i = W(s, r, i)$, and $h_j = W(s, r, j)$, then it easy to determine s from z, i, j, h_i, h_j whenever $i \neq j$.*
4. *(Checkable) If $z = e(s, r)$, then it is easy to determine from z, h, i whether they are consistent (i.e., whether there exist s, r such that $z = e(s, r)$ while $h = W(s, r, i)$).*

6.4.2 Heuristic Implementation of Embedding Scheme

The cut-and-choose scheme of Chaum, Fiat, and Naor [43] is a heuristic embedding scheme. Let f and g be two-argument collision-free functions, such that f is “similar to a random oracle”, and such that g is one-to-one when its first argument is fixed. A secret s is hidden as $e(s, r) = f(x_1, y_1), \dots, f(x_k, y_k)$ where $x_i = g(a_i, b_i), y_i = g(a_i \oplus s, c_i)$ for each $1 \leq i \leq k$; here all a_i, b_i, c_i are determined from the random input r . The witness function W is given by $W(s; r, \gamma_k \dots \gamma_1) = [u_1, v_1, w_1, \dots, u_k, v_k, w_k]$. Here $u_i = g(a_i, b_i), v_i = a_i \oplus s, w_i = c_i$ when $\gamma_i = 0$, and $u_i = a_i, v_i = b_i, w_i = g(a_i \oplus s, c_i)$ when $\gamma_i = 1$.

In a sense, each of the k components is “de-nested” in a particular way, revealing the lowest level of one half, and the intermediate level of the other half. Two distinct witnesses gives the lowest levels of both halves of some component, from which the secret s can be recovered. The properties of an Embedding scheme can be seen to hold heuristically (but not formally, due in part to the difficulty of formalizing the required properties of f and g).

6.4.3 Provably Secure Embedding Scheme

In this section, we present an implementation of an embedding scheme that is provably secure under the Discrete Log Assumption. Intuitively, the secret is given by a set of lines whose slopes encode the secret. One hint gives a point on each line; two hints allows all of the lines to be recovered through interpolation. Exponentiation of the line coefficients modulo a large prime both hides the coefficients (unless discrete logs can be easily computed), and enables anyone to easily verify the incidence of points on lines (thanks to the laws of exponents). Subsequent to our work, other off-line coin schemes have also used line-based embedding schemes [71] [32].

Theorem 6.2 *Under the Discrete Log Assumption, there exists a secure Embedding scheme*

Proof : Let p be a large prime such that $p - 1$ has one large factor q , and let $g \in Z_p$. Let H be a hard bit for the discrete log. Define $e(s, r)$ to be $[g^{a_1} \bmod p, g^{b_1} \bmod p, \dots, g^{a_{|s|}} \bmod p, g^{b_{|s|}} \bmod p]$ where $s[i] = H(g^{a_i} \bmod p)$ for all i , $a_i < q$ for each i , and each a_i, b_i is otherwise random (given by, say, bits $r[(i-1)(|p|+|q|-1)+1, \dots, i(|p|+|q|-1)]$ of r). Define $W(s, r, i)$ to be $[a_1 i + b_1 \bmod p - 1, \dots, a_{|s|} i + b_{|s|} \bmod p - 1]$. We show that the four requirements of an Embedding scheme are satisfied:

1. (Pseudo-Injective) This is trivially satisfied, since e is injective.
2. (Once-Concealed) Suppose that information about $s_i = H(g^{a_i} \bmod p)$ could be determined with probability better than random guessing from $g^{a_i} \bmod p, g^{b_i} \bmod p, i, a_i i + b_i \bmod p - 1$. Then H is not a hard bit for the discrete log, since $H(g^x \bmod p)$ could thus be determined from $g^x \bmod p$ better than random guessing by converting it into the former problem: $g^x \bmod p, (g^x \bmod p)^{-i} g^r \bmod p, i, r$ for random $r \in Z_{p-1}$.
3. (Twice-Revealed) For each $1 \leq l \leq |s|$, interpolate $(i, a_l i + b_l \bmod p - 1)$ and $(j, a_l j + b_l \bmod p - 1)$ to recover a line such that the slope is less than q . Each such line is uniquely determined unless $i - j$ is a multiple of q (which is impossible for $i \neq j$ when $0 < i, j < q$). The hard bits of the slopes yield the bits of s , and the other bits of the coefficients reveal the bits of r .

4. (Checkable) Given $z = [z_{11}, z_{12}, \dots, z_{|s|,1}, z_{|s|,2}], h_1, \dots, h_{|s|}, i$, the check for consistency is whether $z_{l,1}^i z_{l,2} = g^{h_l} \pmod p$ for each $1 \leq l \leq |s|$.

□

6.5 Off-Line Coin Scheme

Informally, an electronic coin scheme [39] is the simplest useful electronic payment system. Any customer can withdraw a coin from the bank, as the bank debits the customer's account. That coin can then be used to purchase from any vendor. The vendor will be able to trust that the coin is genuine. A vendor can later deposit a coin, and the bank will credit the vendor's account. The rights and privileges of customers, vendors, and the bank are all protected within the scheme. The customer is guaranteed anonymity of purchase, i.e., that the bank cannot link a deposited coin to its corresponding withdrawal. The vendor is guaranteed that any acceptable purchase will lead to an acceptable deposit. The bank is guaranteed that no number of withdrawals can lead to a greater number of deposits.

In an off-line coin scheme [43], the bank is not contacted during purchases. Thus protection against the same coin being reused with different vendors can only be guaranteed indirectly. Repeated purchases with the same coin will be accepted initially, but the customer will be caught after the vendors make their deposits. From two deposited versions of the same coin, the identity of the customer can be efficiently extracted (and the customer then penalized appropriately). One of the main difficulties of constructing an off-line coin scheme is to balance this need for reuse exposure with the need for anonymity of purchase.

6.5.1 Definition of Secure Off-Line Coins

There is a bank B , a collection of vendors $\{V_j\}$, and a collection of customers $\{C_i\}$, all of whom are assumed to be communicating Turing Machines. There is a security parameter k .

An off-line coin scheme consists of three protocols: a withdrawal protocol between B and any C_i ; a purchase protocol between any C_i and any V_j ; and a deposit protocol between any V_j and B . The withdrawal protocol begins with C_i and B having id_i on their shared input tape, and ends with C_i having a "coin" τ on its private output tape. The purchase protocol begins with C_i having a coin τ on its input tape, and ends with V_j having a "spent coin" τ' on its private output tape. The deposit protocol begins with V_j having a spent coin τ' on its input tape and ends with B having a "deposited coin" τ'' on its private output tape.

We capture the security of an off-line coin scheme in four crisp requirements. Unreusability means that the same coin cannot be spent twice without revealing the identity of the customer. Unexpandability means that more coins can never be constructed from fewer withdrawals. Unforgeability means that no conspiracy can create a coin without having the identity of one of its members embedded within it. Untraceability is stated strongly, in that no purchase can be linked by the bank to its corresponding withdrawal even if somehow narrowed down to only two possible withdrawal instances (and by unforgeability, every coin can be traced to someone).

Definition 6.3 *An off-line coin scheme is secure if the following requirements are satisfied:*

1. (Unreusable) If any C_i begins two successful purchase protocols with the same coin τ on its input tape, then the fact of reuse and the identity id_i can be computed in p.p.t. from the two corresponding deposited coins τ_1'', τ_2'' , except with negligible probability (in k).
2. (Unexpandable) From the views of the customers of n withdrawal protocols, no p.p.t. Turing Machine can compute $n + 1$ distinct coins that will lead to successful purchase protocols (or $n + 1$ distinct spent coins that will lead to successful deposit protocols), except with negligible probability (in k).
3. (Unforgeable) If, from the views of the customers of n withdrawal protocols, some p.p.t. Turing Machine can compute a single coin that will lead to two successful purchase protocols, then the fact of reuse and the identity of at least one of these customers can be efficiently computed from the two corresponding deposited coins, except with negligible probability (in k).
4. (Untraceable) Let V_B^i (V_V^i) be the view of B (V) for a withdrawal (purchase) protocol with C_i . Then, for all i, j , no p.p.t Turing Machine on input V_B^i, V_B^j , and $V_V^l \in_R \{V_V^i, v_V^j\}$, can output a guess for $l \in \{i, j\}$ non-negligibly better than random guessing (in k).

6.5.2 Provably Secure Off-Line Coin Scheme

In this section, we justify our focus on Oblivious Authentication and Embedding schemes. The following theorem establishes a connection between these primitives and secure off-line coins.

Theorem 6.3 *A secure off-line coin scheme is possible given an Oblivious Authentication scheme $[R, A, e]$ such that e is part of an Embedding scheme e, W .*

Proof :

We can show that the following coin scheme is secure:

- The Withdrawal protocol is an instance of the Oblivious Authentication scheme with the identity id_C of the Customer as the shared string. The Customer gets r, σ at the end of the O.A. protocol, and can compute the coin z, σ where $z = e(s, r)$.
 - The Customer begins the Purchase protocol by sending z, σ to the Vendor. The Vendor responds by sending a random challenge i to the Customer. The Customer responds by sending h_i to the Vendor, where $h_i = W(s, r, i)$. The Vendor accepts the purchase if σ is a valid signature of z under the Bank's secret key, and if z, h_i are consistent.
 - For the Deposit protocol, the Vendor sends z, σ, i, h_i to the Bank.
1. (Unreusable) Suppose that the two purchases leave the Vendor with z, σ, h'_i and z, σ, h'_j as responses to his challenges i and j respectively. The fact of reuse can be detected by the Bank after both coins are deposited. By twice-revealability of the Embedding scheme, then, the Bank will learn the identity of a double spender if both h'_i and h'_j are computed correctly. If at least one of these is computed incorrectly, then One-Wayness of the Embedding scheme would be violated.

2. (Unexpandable) Violation of unexpandability for off-line coins implies a violation of unexpandability for Oblivious Authentication. This is because the Vendor will reject any purchase using a coin $[z, \sigma]$ unless σ is a valid signature of z . Thus $n + 1$ successful purchases with distinct coins implies $n + 1$ valid signatures, which cannot be obtained from n O.A. executions.
3. (Unforgeable) Violation of unforgeability for off-line coins implies a violation of unforgeability for Oblivious Authentication. The Vendor rejects any purchase unless it uses a coin $[z, \sigma]$ where σ is a valid signature of z . This implies that $z = e(s_i, r_i)$ where s_i was the shared secret for one of the (Withdrawal) O.A. executions. Double spending will cause s_i to be revealed, which will tell the Bank the identity of the Customer.
4. (Untraceable) Let M be a p.p.t. TM with input V_B^i, V_B^j, V_l^V that outputs a guess for l with success non-negligibly better than random guessing. Let s_i, s_j be the shared secrets for the two Withdrawal protocols, and let k, h_k be the hint of the witness given in the Purchase protocol. By the obliviousness of O.A., there exists a p.p.t. TM M' with input $s_i, s_j, z, \sigma, k, h_k$ that guesses l with negligibly close success (viewing k, h_k as auxiliary input to M and M'). This violates the once-concealed property of the embedding scheme, since M' , with the private signature key of B hard-wired in, can be used to choose from $\{s_i, s_j\}$ given s_i, s_j, z, k, h_k with a non-negligible advantage over random guessing.

□

6.6 Practical Implications

We have shown that any Embedding scheme, together with any Oblivious Authentication scheme, implies the existence of a secure off-line coin scheme. For each of Embedding and Oblivious Authentication, we have shown a new implementation that is provably secure with respect to cryptographic assumptions, and pointed out existing implementations that are heuristically but not provably sound.

How practical are our methods? If both of our provably secure implementations are used, then the resulting off-line coin scheme is efficient in the sense described in Section 6.1. The communication complexity is a low-degree polynomial of the relevant parameters. Specifically, the size of the coin $[z, \sigma]$ is $O(mk)$ signed elements; this is also the communication complexity of the protocol. By hashing the components of z , m can be reduced to $O(k)$, for a total complexity of $O(k^2s)$ bits, where s is the size of a signature (using the underlying signature scheme *sig*) of a single bit. If a trusted manufacturer is present, then the pre-processing stage for each customer consists of a single transfer of structs, upon request, from the trusted manufacturer to the customer (or more frequent transfers, upon request, as needed); the manufacturer does not need to know the identity of the customer for this pre-processing stage. If a general secure 2-party computation protocol were to be used between the Bank and Customer, then the pre-processing stage (independent of any money related activities) would have round and message complexities that are polynomial in the size of the circuit to compute a struct from jointly produced random values.

If our provably secure Embedding scheme (Section 6.4.3) is combined with the heuristic Oblivious Authentication (Section 6.3.2), then the resulting scheme can be quite practical. Assume that, instead of a single hard bit, there are as many hard bits in the discrete log as the size of the id of a

customer. In this case, each z_i is only $2 \log p$ bits long (hiding the coefficients of a single line, whose slope encodes all of the identity of the Customer), and the authenticating string is only $\log N_A$ bits long. Thus the total length of a coin at purchase or deposit time is only $O(\log N_A + k \log p)$ bits (and encryption complexity is $O(k)$ modular exponentiations), which is quite reasonable. Withdrawal requires only two rounds of interaction, while purchase and deposit are non-interactive.

This is an example of taking a secure scheme (for off-line coins) that is “efficient” in the sense discussed in the Introduction, and plugging in a heuristic implementation (for Oblivious Authentication). The low complexity of the resulting practical scheme helps to demonstrate how our notion of efficiency of protocols coincides with practical efficiency.

Chapter 7

Conclusion

The research described in this thesis touches on a number of topics in the general area of secure distributed computation. We have studied new computational and adversarial models, and established new connections to other areas of research (e.g., graph theory). One unifying thread has been the focus on efficiency, especially of communication resources. For problems which already had general theoretical solutions, we have taken a closer look, establishing lower bounds, tradeoffs, and reductions of important resources. This common focus is not accidental. We have been guided throughout this work by the observation that a gap separated the elegant theoretical contributions of past researchers from practical applicability. Our work has attacked this gap, but not closed it. We believe that many practical benefits of secure distributed computation are possible, but they will not be realized without further theoretical work. In addition to the many open problems mentioned in the body of this thesis, we close with two additional challenges of particular interest.

In Section 5, we showed how a small (constant factor) gain in bit complexity was possible for the comparison of bit strings. Comparison is a useful function in practice, e.g., for financial transactions such as sealed-bid auctions. Further exploration is needed to see whether simple and practical special purpose protocols can be designed for useful distributed tasks. Plausible candidates for domain areas include distributed relational databases, automated off-hours trading systems, sensor fusion, and numerical modeling. For many problems in these domain areas, the actual data operations are quite simple. Perhaps some new approaches can meet security needs without the elaborate machinery developed for general secure computation.

The second research direction concerns the ways that secure protocols can be combined. We can motivate this by an example. Suppose that we have designed a special-purpose secure protocol for conducting a sealed-bid auction. The inputs to this protocol are the secret bids, and the output is a determination of the winner. What should happen at the end of a secure auction? Presumably, the winners are contacted and asked to pay the appropriate amounts. What if a winner refuses to pay? One answer is that this is outside the scope of the protocol design. Another answer is that other protocols could perhaps be “folded” into the auction protocol to address the problem. In particular, it may be possible to fold an electronic money scheme into an auction protocol so that every participant has “paid in advance” (potentially). The digital money of winners would somehow be activated by the end of the auction, while losers were assured that their money remained unusable. It seems promising to explore the proper ways to combine different protocols to provide seamless secure services.

Bibliography

- [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] C. van Antwerpen, “Electronic cash,” Master’s thesis, Eindhoven University of Technology, 1990.
- [4] 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.
- [5] I. Barany and Z. Furedi, “Mental poker with three or more players,” *Information and Control* 59 (1983), 84–93.
- [6] J. Bar-Ilan and D. Beaver, “Non-cryptographic fault-tolerant computing in a constant number of rounds of interaction,” *PODC 1989*, 201–209.
- [7] D. Barrington, “Bounded-width branching programs recognize exactly those languages in NC^1 ,” *J. Comput. System Sci.* 38 (1989), 150–164.
- [8] R. Bar-Yehuda, B. Chor, and E. Kushilevitz, “Privacy, additional information, and communication,” *IEEE Structure in Complexity Theory 1990*, 55–65.
- [9] D. Beaver, “Multiparty protocols tolerating half faulty processors,” in *Advances in Cryptology—CRYPTO ’89 Proceedings* (Lecture Notes in Computer Science, Vol. 435), ed. G. Brassard, 560–572, Springer-Verlag, New York, 1990.
- [10] D. Beaver, “Perfect privacy for two-party protocols,” *DIMACS Workshop on Distributed Computing and Cryptography*, Feigenbaum and Merritt (eds.), AMS, 1990, 65–77.
- [11] D. Beaver, “Foundations of secure interactive computing,” in *Advances in Cryptology—CRYPTO ’91 Proceedings* (Lecture Notes in Computer Science, Vol. 576), ed. J. Feigenbaum, 377–391, Springer-Verlag, New York, 1992.
- [12] D. Beaver, “Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority,” *J. Cryptology* (1991) 4: 75–122.
- [13] D. Beaver and J. Feigenbaum, “Hiding instances in multioracle queries,” *STACS 1990*, 37–48.

- [14] D. Beaver, J. Feigenbaum, J. Kilian, and P. Rogaway, “Security with low communication overhead,” in *Advances in Cryptology—CRYPTO ’90 Proceedings* (Lecture Notes in Computer Science, Vol. 537), ed. A. Menzes and S. Vanstone, 62–76, Springer-Verlag, New York, 1991.
- [15] D. Beaver and S. Goldwasser, “Multiparty computation with faulty majority,” IEEE FOCS 1989, 468–473.
- [16] D. Beaver, S. Micali, and P. Rogaway, “The round complexity of secure protocols,” ACM STOC 1990, 503–513.
- [17] 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.
- [18] J. Benaloh (Cohen), “Secret sharing homomorphisms: keeping shares of a secret secret,” in *Advances in Cryptology—CRYPTO ’86 Proceedings* (Lecture Notes in Computer Science, Vol. 263), ed. A. Odlyzko, 251–260, Springer-Verlag, New York, 1987.
- [19] J. Benaloh (Cohen) and M. Yung, “Distributing the power of a government to enhance to privacy of voters,” PODC 1986, 52–62.
- [20] M. Ben-Or, R. Canetti, and O. Goldreich, “Asynchronous secure computation,” ACM STOC 1993, 52–61.
- [21] M. Ben-Or and R. Cleve, “Computing algebraic formulas using a constant number of registers,” ACM STOC 1988, 254–257.
- [22] M. Ben-Or, S. Goldwasser, and A. Wigderson, “Completeness theorems for non-cryptographic fault-tolerant distributed computation,” ACM STOC 1988, 1–9.
- [23] R. Berger, R. Peralta, and T. Tedrick, “A provably secure oblivious transfer protocol,” Eurocrypt 1984, 379–386.
- [24] E. Berlekamp, *Algebraic Coding Theory*, Aegean Park Press, Laguna Hills, CA, 1984.
- [25] D. Bienstock and P. Seymour, “Monotonicity in graph searching,” J. Algorithms 12 (1991), 230–245.
- [26] 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.
- [27] M. Blum, “Coin flipping by telephone: a protocol for solving impossible problems,” IEEE Computer Conference 1982, 133–137.
- [28] M. Blum, “How to exchange (secret) keys,” ACM Trans. Comput. Sys. 1 (1983), 175–193.
- [29] 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.

- [30] M. Blum and S. Micali, “How to generate cryptographically strong sequences of pseudo random bits”, *SIAM J. Comput.* 13 (1984), 850–864.
- [31] M. Blum, U. Vazirani, and V. Vazirani, “Reducibility among protocols,” in *Advances in Cryptology—CRYPTO ’83 Proceedings*, 137–146, Plenum Press, 1984.
- [32] S. Brands, “Electronic cash systems based on the representation problem in groups of prime order,” in *Advances in Cryptology—CRYPTO ’93 Proceedings* (Lecture Notes in Computer Science), ed. D. Stinson, Springer-Verlag, New York (to appear).
- [33] G. Brassard, D. Chaum, and C. Crépeau, “Minimum disclosure proofs of knowledge,” *J. Comput. System Sci.* 37 (1988) 156–189.
- [34] G. Brassard, C. Crépeau, and J. Robert, “Information theoretic reductions among disclosure problems,” *IEEE FOCS* 1986, 168–173.
- [35] G. Brassard, C. Crépeau, and M. Yung, “Perfectly concealing computationally convincing interactive proofs in constant rounds,” *Theoretical Computer Science* (to appear).
- [36] R. Breisch, “An intuitive approach to speleo-topology,” *Southwestern Cavers* (published by the Southwestern Region of the National Speleological Society) 6 (1967), 72–78.
- [37] M. Burrows, M. Abadi, and R. Needham, “Authentication: A practical study in belief and action,” in *Proceedings of the Second Conference on Theoretical Aspects of Reasoning about Knowledge*, Moshe Vardi (ed.), Morgan Kaufmann, 1988.
- [38] D. Chaum, “Untraceable electronic mail, return addresses and digital pseudonyms,” *CACM* 24 (1981), 84–88.
- [39] D. Chaum, “Security without identification: transaction systems to make big brother obsolete,” *CACM* 28, 10 (October 1985).
- [40] D. Chaum, “The spymasters double-agent problem: multiparty computations secure unconditionally from minorities and cryptographically from majorities,” in *Advances in Cryptology—CRYPTO ’89 Proceedings* (Lecture Notes in Computer Science, Vol. 435), ed. G. Brassard, 591–601, Springer-Verlag, New York, 1990.
- [41] D. Chaum, C. Crépeau, and I. Damgård, “Multiparty unconditionally secure protocols,” *ACM STOC* 1988, 11–19.
- [42] D. Chaum, I. Damgård, and J. van de Graaf, “Multiparty computations ensuring privacy of each party’s input and correctness of the result,” in *Advances in Cryptology—CRYPTO ’87 Proceedings* (Lecture Notes in Computer Science, Vol. 293), ed. C. Pomerance, 87–119, Springer-Verlag, New York, 1988.
- [43] D. Chaum, A. Fiat, and M. Naor, “Untraceable electronic cash,” in *Advances in Cryptology—CRYPTO ’88 Proceedings* (Lecture Notes in Computer Science, Vol. 403), ed. S. Goldwasser, 319–327, Springer-Verlag, New York, 1990.

- [44] D. Chaum and T. Pederson, “Wallet databases with observers,” in *Advances in Cryptology—CRYPTO ’92 Proceedings* (Lecture Notes in Computer Science), ed. E. Brickell, 90–106, Springer-Verlag, New York, 1993.
- [45] B. Chor, M. Geraud-Graus, and E. Kushilevitz, “Private computations over the integers,” *IEEE FOCS 1990*, 335–344.
- [46] 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.
- [47] B. Chor and E. Kushilevitz, “A zero-one law for boolean privacy,” *ACM STOC 1989*, 62–72.
- [48] R. Cleve, “Limits on the security of coin flips when half the processors are faulty,” *ACM STOC 1986*, 364–369.
- [49] R. Cleve, “Controlled gradual disclosure schemes for random bits and their applications,” in *Advances in Cryptology—CRYPTO ’89 Proceedings* (Lecture Notes in Computer Science, Vol. 435), ed. G. Brassard, 573–588, Springer-Verlag, New York, 1990.
- [50] J. (Benaloh) Cohen and M. Fisher, “A robust and verifiable cryptographically secure election scheme,” *IEEE FOCS 1985*, 372–382.
- [51] D. Coppersmith, “Cheating at mental poker,” in *Advances in Cryptology—CRYPTO ’85 Proceedings* (Lecture Notes in Computer Science, Vol. 218), ed. H. Williams, 104–107, Springer-Verlag, New York, 1986.
- [52] C. Crépeau, “A secure poker protocol that minimizes the effect of player coalitions,” in *Advances in Cryptology—CRYPTO ’85 Proceedings* (Lecture Notes in Computer Science, Vol. 218), ed. H. Williams, 73–86, Springer-Verlag, New York, 1986.
- [53] C. Crépeau, “A zero-knowledge poker protocol that achieves confidentiality of the players’ strategy, or How to achieve an electronic poker face,” in *Advances in Cryptology—CRYPTO ’86 Proceedings* (Lecture Notes in Computer Science, Vol. 263), ed. A. Odlyzko, 239–247, Springer-Verlag, New York, 1987.
- [54] C. Crépeau, “Equivalence between two flavours of Oblivious Transfer,” in *Advances in Cryptology—CRYPTO ’87 Proceedings* (Lecture Notes in Computer Science, Vol. 293), ed. C. Pomerance, 350–354, Springer-Verlag, New York, 1988.
- [55] C. Crépeau and J. Kilian, “Achieving oblivious transfer using weakened security assumptions,” *IEEE FOCS 1988*, 42–52.
- [56] I. Damgård, “Payment systems and credential mechanisms with provable security against abuse by individuals,” in *Advances in Cryptology—CRYPTO ’88 Proceedings* (Lecture Notes in Computer Science, Vol. 403), ed. S. Goldwasser, 328–335, Springer-Verlag, New York, 1990.
- [57] A. DeSantis and G. Persiano, “Communication efficient zero-knowledge proofs of knowledge (with applications to electronic cash),” *STACS 1992*, 449–460.

- [58] Y. Desmedt, “Society and group oriented cryptography: A new concept,” in *Advances in Cryptology—CRYPTO ’87 Proceedings* (Lecture Notes in Computer Science, Vol. 293), ed. C. Pomerance, 120–127, Springer-Verlag, New York, 1988.
- [59] Y. Desmedt and Y. Frankel, “Threshold cryptosystems,” in *Advances in Cryptology—CRYPTO ’89 Proceedings* (Lecture Notes in Computer Science, Vol. 435), ed. G. Brassard, 307–315, Springer-Verlag, New York, 1990.
- [60] W. Diffie and M. Hellman, “New directions in cryptography,” *IEEE Transactions on Information Theory*, 22(6): 644–654, 1976.
- [61] D. Dolev, C. Dwork, O. Waarts, and M. Yung, “Perfectly secure message transmission,” *JACM* 40 (1993), 17–47.
- [62] D. Dolev and A. Yao, “On the security of public key protocols,” *ACM FOCS 1981*, 350–357.
- [63] J. Edmonds and R. Karp, “Theoretical improvements in algorithmic efficiency for network flow problems,” *JACM* 19 (1972), 248–264.
- [64] T. El-Gamal, “A public key cryptosystem and a signature scheme based on discrete logarithms,” *IEEE Transactions on Information Theory*, 31: 469–472, 1985.
- [65] S. Even, O. Goldreich, and A. Lempel, “A randomized protocol for signing contracts,” *CACM* 28 (1985), 637–647.
- [66] S. Even, O. Goldreich, and S. Micali, “On-line/off-line digital signatures,” in *Advances in Cryptology—CRYPTO ’89 Proceedings* (Lecture Notes in Computer Science, Vol. 435), ed. G. Brassard, 263–275, Springer-Verlag, New York, 1990.
- [67] S. Even, O. Goldreich, and Y. Yacobi, “Electronic Wallet,” in *Advances in Cryptology—CRYPTO ’83 Proceedings*, 383–386, Plenum Press, 1984.
- [68] T. Feder, E. Kushilevitz, and M. Naor, “Amortized communication complexity,” *IEEE FOCS 1991*, 239–248.
- [69] U. Feige, A. Fiat, and A. Shamir, “Zero-Knowledge Proofs of Identity,” *J. Cryptology* 1 (1988) 77–94.
- [70] P. Feldman and S. Micali, “Optimal algorithms for Byzantine agreement,” *ACM STOC 1988*, 148–161.
- [71] N. Ferguson, “Extensions of single-term coins,” in *Advances in Cryptology—CRYPTO ’93 Proceedings* (Lecture Notes in Computer Science), ed. D. Stinson, Springer-Verlag, New York (to appear).
- [72] A. Fiat and A. Shamir, “How to prove yourself: Practical solutions to identification and signature problems,” in *Advances in Cryptology—CRYPTO ’86 Proceedings* (Lecture Notes in Computer Science, Vol. 263), ed. A. Odlyzko, 186–194, Springer-Verlag, New York, 1987.

- [73] S. Fortune and M. Merritt, “Poker protocols,” in *Advances in Cryptology—CRYPTO ’84 Proceedings* (Lecture Notes in Computer Science), 454–464, Springer-Verlag, New York, 1985.
- [74] M. Franklin, Z. Galil, and M. Yung, “Eavesdropping games: A graph-theoretic approach to privacy in distributed systems,” *IEEE FOCS 1993*, 670-679.
- [75] M. Franklin and S. Haber, “Joint encryption and message-efficient secure computation,” in *Advances in Cryptology—CRYPTO ’93 Proceedings* (Lecture Notes in Computer Science), ed. D. Stinson, Springer-Verlag, New York (to appear).
- [76] M. Franklin and M. Yung, “Communication complexity of secure computation,” *ACM STOC 1992*, 699-710.
- [77] M. Franklin and M. Yung, “Secure and efficient off-line digital money,” *Proc. 20th International Colloquium on Automata, Languages and Programming (ICALP 1993)*, Lund, Sweden, July 1993, *Lecture Notes in Computer Science 700*, Springer-Verlag, Berlin, 1993, 265-276.
- [78] Z. Galil, S. Haber, and M. Yung, “Cryptographic computation: secure fault-tolerant protocols and the public-key model,” in *Advances in Cryptology—CRYPTO ’87 Proceedings* (Lecture Notes in Computer Science, Vol. 293), ed. C. Pomerance, 135–155, Springer-Verlag, New York, 1988.
- [79] M. Garey, D. Johnson, and L. Stockmeyer, “Some simplified NP-complete graph problems,” *Theoretical Computer Science 1* (1976), 237–267.
- [80] O. Goldreich and L. Levin, “A hard-core predicate for all one-way functions,” *STOC 1989*, 25–32.
- [81] 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.
- [82] O. Goldreich, S. Micali, and A. Wigderson, “How to play any mental game,” *ACM STOC 1987*, 218–229.
- [83] O. Goldreich and R. Vainish, “How to solve any protocol problem – an efficiency improvement,” in *Advances in Cryptology—CRYPTO ’87 Proceedings* (Lecture Notes in Computer Science, Vol. 293), ed. C. Pomerance, 73–86, Springer-Verlag, New York, 1988.
- [84] S. Goldwasser and L. Levin, “Fair computation of general functions in presence of immoral majority,” in *Advances in Cryptology—CRYPTO ’89 Proceedings* (Lecture Notes in Computer Science, Vol. 435), ed. G. Brassard, 75–84, Springer-Verlag, New York, 1990.
- [85] S. Goldwasser and S. Micali, “Probabilistic encryption,” *J. Comput. System Sci.* 28 (1984), 270–299.
- [86] S. Goldwasser, S. Micali, and C. Rackoff, “The knowledge complexity of interactive proof systems,” *SIAM J. Comput.* 18 (1989), 186–208.
- [87] S. Haber, “Multiparty cryptographic computation: techniques and applications,” Ph.D. thesis, Columbia University, 1988.

- [88] J.T. Hastad, “Pseudo-random generators under uniform assumptions,” STOC 1990, 395–404.
- [89] J. Håstad and A. Shamir, “The cryptographic security of truncated linearly related variables,” STOC 1985, 356–362.
- [90] R. Hirschfeld, “Making electronic refunds safer,” in *Advances in Cryptology—CRYPTO ’92 Proceedings* (Lecture Notes in Computer Science), ed. E. Brickell, 107–113, Springer-Verlag, New York, 1993.
- [91] M. Huang and S. Teng, “Security, verifiability, and universality in distributed computing,” *J. Algorithms* 11 (1990), 492–521.
- [92] R. Impagliazzo, L. Levin, and M. Luby, “Pseudorandom number generation from one-way functions,” *ACM STOC* 1989, 12–24.
- [93] R. Impagliazzo and M. Luby, “One-way functions are essential for complexity based cryptography,” *IEEE FOCS* 1989, 230–235.
- [94] R. Impagliazzo and S. Rudich, “Limits on the provable consequences of one-way permutations,” *ACM STOC* 1989, 44–61.
- [95] R. Impagliazzo, and M. Yung, “Direct minimum-knowledge computation,” in *Advances in Cryptology—CRYPTO ’87 Proceedings* (Lecture Notes in Computer Science, Vol. 293), ed. C. Pomerance, 40–51, Springer-Verlag, New York, 1988.
- [96] J. Kilian, “Founding cryptography on oblivious transfer,” *ACM STOC* 1988, 20–31.
- [97] J. Kilian, “Uses of Randomness in Algorithms and Protocols,” *ACM Distinguished Dissertation*, MIT Press, 1990.
- [98] J. Kilian, “A general completeness theorem for two-party games,” *ACM STOC* 1991, 553–560.
- [99] L. Kirousis and C. Papadimitriou, “Interval graphs and searching,” *Discrete Mathematics* 55 (1985) 181–184.
- [100] L. Kirousis and C. Papadimitriou, “Searching and pebbling,” *Theoretical Computer Science* 47 (1986), 205–218.
- [101] E. Kushilevitz, “Privacy and communication complexity,” *IEEE FOCS* 1989, 416–421.
- [102] E. Kushilevitz, “On computing the sum privately,” manuscript.
- [103] L. Lamport, R. Shostak, and M. Pease, “The Byzantine generals problem,” *ACM Trans. on Programming Lang. and Systems* (1982), 382–401.
- [104] A. LaPaugh, “Recontamination does not help to search a graph,” *JACM*, April 1993 (originally Princeton Technical Report 335).
- [105] R. Lipton, “How to cheat at mental poker,” proceedings of AMS short course on cryptography, 1981.

- [106] T. Long and A. Wigderson, “The discrete logarithm hides $O(\log n)$ bits,” *SIAM J. Comput.* 17 (1988), 363–372.
- [107] 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.
- [108] F. Makedon, C. Papadimitriou, and I. Sudborough, “Topological bandwidth,” *SIAM J. Alg. Disc. Meth.* 6 (1985), 418–444.
- [109] K. McCurley, “A key distribution system equivalent to factoring,” *J. Cryptology*, 1 (1988), 95–105.
- [110] N. Megiddo, S. Hakimi, M. Garey, D. Johnson, and C. Papadimitriou, “The complexity of searching a graph,” *JACM* 35 (1988), 18–44.
- [111] M. Merritt, “Cryptographic protocols,” Ph.D. thesis, Georgia Institute of Technology, 1983.
- [112] S. Micali, “Fair public-key cryptosystems,” in *Advances in Cryptology—CRYPTO ’92 Proceedings* (Lecture Notes in Computer Science), ed. E. Brickell, 114–139, Springer-Verlag, New York, 1993.
- [113] S. Micali and P. Rogaway, “Secure computation,” in *Advances in Cryptology—CRYPTO ’91 Proceedings* (Lecture Notes in Computer Science, Vol. 576), ed. J. Feigenbaum, 392–404, Springer-Verlag, New York, 1992.
- [114] M. Naor, “Bit commitment using pseudo-randomness,” in *Advances in Cryptology—CRYPTO ’89 Proceedings* (Lecture Notes in Computer Science, Vol. 435), ed. G. Brassard, 128–136, Springer-Verlag, New York, 1990.
- [115] M. Naor, R. Ostrovsky, R. Venkatesan, and M. Yung, “Perfect zero-knowledge arguments for NP can be based on general complexity assumptions,” manuscript, 1991.
- [116] M. Naor and M. Yung, “Universal one-way hash functions and their cryptographic applications,” *STOC 1989*, 33–43.
- [117] T. Okamoto and K. Ohta, “Disposable zero-knowledge authentications and their applications to untraceable electronic cash,” in *Advances in Cryptology—CRYPTO ’89 Proceedings* (Lecture Notes in Computer Science, Vol. 435), ed. G. Brassard, 481–496, Springer-Verlag, New York, 1990.
- [118] T. Okamoto and K. Ohta, “Universal electronic cash,” in *Advances in Cryptology—CRYPTO ’91 Proceedings* (Lecture Notes in Computer Science, Vol. 576), ed. J. Feigenbaum, 324–337, Springer-Verlag, New York, 1992.
- [119] A. Orlitsky and A. El Gamal, “Communication with secrecy constraints,” *ACM STOC 1984*, 217–224.
- [120] R. Ostrovsky, R. Venkatesan, and M. Yung, “Fair games against an all-powerful adversary,” *Sequences Workshop, Positano, Italy, July 1991*.

- [121] 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.
- [122] R. Ostrovsky and M. Yung, “Robust computation in the presence of mobile viruses,” ACM PODC 1991, 51–59.
- [123] T. Parsons, “Pursuit-evasion in a graph,” in “Theory and application of graphs,” (Y. Alavi and D. Lick, eds.), Springer-Verlag (1976), 426–441.
- [124] B. Pfitzmann and M. Waidner, “How to break and repair a ‘provably secure’ untraceable payment system,” in *Advances in Cryptology—CRYPTO ’91 Proceedings* (Lecture Notes in Computer Science, Vol. 576), ed. J. Feigenbaum, 338–350, Springer-Verlag, New York, 1992.
- [125] M. Rabin, “Digital signatures,” in Foundations of Secure Computation, R. DeMillo, D. Dobkin, A. Jones, and R. Lipton (editors), Academic Press, NY, 1978, 155–168.
- [126] M. Rabin, “How to exchange secrets by oblivious transfer,” Tech. Memo TR-81, Aiken Computation Laboratory, Harvard University, 1981.
- [127] T. Rabin, “Robust sharing of secrets when the dealer is honest or cheating,” M.Sc. Thesis, Hebrew University, 1988.
- [128] T. Rabin and M. Ben-Or, “Verifiable secret sharing and multiparty protocols with honest majority,” ACM STOC 1989, 73–85.
- [129] R. Rivest, A. Shamir, and L. Adleman, “A method for obtaining digital signatures and public key cryptosystems,” CACM 21 (1978), 120–126.
- [130] J. Rompel, “One-way functions are necessary and sufficient for secure signatures,” STOC 1990, 387–394.
- [131] J. Schwartz, “Fast probabilistic algorithms for verification of polynomial identities,” JACM 27 (1980), 701–717.
- [132] A. Shamir, “How to share a secret,” CACM 22 (1979), 612–613.
- [133] A. Shamir, R. Rivest, and L. Adleman, “Mental poker,” Technical Report MIT/LCS/TR-125, M.I.T., 1979.
- [134] G. Simmons, “Geometric shared secret and/or shared control schemes,” in *Advances in Cryptology—CRYPTO ’90 Proceedings* (Lecture Notes in Computer Science, Vol. 537), ed. A. Menzes and S. Vanstone, 216–240, Springer-Verlag, New York, 1991.
- [135] C. Small, *Arithmetic of Finite Fields*, Monographs and Textbooks in Pure and Applied Mathematics (Vol. 148), Marcel Dekker, Inc., New York, 1991.
- [136] M. Tompa and H. Woll, “Random self-reducibility and zero knowledge interactive proofs of possession of information,” IEEE FOCS 1987, 472–482.

- [137] U. Vazirani and V. Vazirani, “Trapdoor pseudo-random number generators, with applications to protocol design,” IEEE FOCS 1983, 23–30.
- [138] A. Yao, “Some complexity questions related to distributive computing,” ACM STOC 1979, 209–213.
- [139] A. Yao, “Protocols for secure computations,” IEEE FOCS 1982, 160–164.
- [140] A. Yao, “How to generate and exchange secrets,” IEEE FOCS 1986, 162–167.
- [141] M. Yung, “Cryptoprotocols: subscription to a public key, the secret blocking and the multi-player mental poker game,” in *Advances in Cryptology—CRYPTO ’84 Proceedings* (Lecture Notes in Computer Science), 439–453, Springer-Verlag, New York, 1985.
- [142] R. Zippel, “Probabilistic algorithms for sparse polynomials,” in *Proc. EUROSAM ’79* (Lecture Notes in Computer Science, Vol. 72), 216–226, Springer-Verlag, 1979.