Abstract

Privacy, Integrity, and Incentive-Compatibility in Computations with Untrusted Parties

Sheng Zhong

2004

In this dissertation, I study privacy, integrity, and incentive compatibility in computations with untrusted parties. The study of privacy and integrity belongs to the research area of *secure multi-party computation*, while incentive compatibility is a natural extension of the research on secure multi-party computation.

First, I present a mix network tailored for election systems, with a substantial speedup over previous work. Second, I design and analyze efficient algorithms for distributed mining of association rules. Third, to protect data integrity in an untrusted storage service, I study the possibility of entangling multiple users' data together in such a way that loss of one user's data implies loss of all others'. Fourth, I introduce VDOT, a new cryptographic primitive, which can be viewed as an extension of oblivious transfer with malicious servers. I also apply VDOT to the problem of mobile-agent security to implement the key components of an architecture for mobile agents.

Finally, I propose a way to add incentive considerations to the study of secure multi-party computation, by stimulating cooperation among selfish mobile nodes in an ad hoc network. I propose Sprite, a simple, cheat-proof, credit-based system for accompanishing this task. The system suppresses cheating behavior and provides incentives for mobile nodes to cooperate and report actions honestly. Simulations and analysis show that mobile nodes can cooperate and forward each other's messages, unless the resources of each node are extremely depleted.

Privacy, Integrity, and Incentive-Compatibility in Computations with Untrusted Parties

A Dissertation
Presented to the Faculty of the Graduate School
of
Yale University
in Candidacy for the Degree of
Doctor of Philosophy

by
Sheng Zhong

Dissertation Director: Joan Feigenbaum

December 2004

Contents

List of Figures

List of Tables

Acknowledgements

I am truly grateful to my advisor, Prof. Joan Feigenbaum, for teaching me how to do research in computer science, especially in computer security. I am also highly indebted to my collaborators —- Prof. James Aspnes, Prof. Dan Boneh, Jiang Chen, Dr. Phillippe Golle, Dr. Markus Jakobsson, Dr. Ari Juels, Aleksandr Yampolskiy, and Prof. Yang Richard Yang. Without their hard work, I would not have been able to complete this dissertation.

I would also like to thank the National Science Foundation (NSF) and the Office of Naval Research (ONR). The research work in this disseration has been supported in part by NSF grant CCR-0208972 and ONR grant N00014-01-1-0795.

Last but not least, I thank my parents, Qiusheng Zhong and Ying Long, and my girlfriend, Yue Ji, so much for their everlasting love and support.

Chapter 1

Introduction

1.1 Background and Motivation

Given the rapid advances of computer and information technology, many security issues need to be addressed. For example, with the unprecedented convenience of accessing information, people are very concerned about the *privacy* of their own data; with the ease of editing information, they are eager to seek protection of the *integrity* of their data. Furthermore, because networking technologies, especially wireless technologies, are connecting huge numbers of mutually unfamiliar people together to make a global economy, the *incentives* of each participant play a key role in designing modern information systems.

In this dissertation, I study privacy, integrity, and incentive compatibility in computations with untrusted parties. Here, by "privacy," I mean the guarantee that certain information is hidden from certain participants. By "integrity," I mean the guarantee that certain data are protected from being corrupted. By incentive compatibility, I mean the guarantee that behaving cooperatively (more specifically, following certain protocols) is the choice of each selfish and rational participant.

Clearly, the study of the first two issues, namely privacy and integrity, in the context of protocol design with multiple mutually untrusted parties belongs to the research area of *secure multi-party computation*. It is my belief that adding the third issue, incentive compatibility, to the research on secure multi-party computation will be essential, because, besides the malicious and semi-honest parties considered in traditional secure multi-party computation, selfish (or economically rational) parties are also very popular in the practice of today's computer networks.

This dissertation includes multiple technical components, which address practical problems in various applications. Each of these problems focuses on one or two of the issues mentioned above and can be viewed as a (in some cases extended) version of secure multi-party computation. For completeness of this dissertation, next I give a brief review of secure multi-party computation.

1.2 Secure Multi-party Computation

In this section, I present a very brief review of secure multi-party computation, listing various models and definitions and some of the important existing results. Due to space limitations, I do not attempt to cover the vast literature on this topic. For more detailed surveys, please see [59, 62]. I also give a comparison of my proposed solutions with the existing solutions to secure multi-party computation and explain why this is still worth studying given the huge amount of existing work in the area.

1.2.1 Models of Secure Multi-party Computation

Secure multi-party computation involves a set of n parties. Each of these parties has a private input, and the goal of the computation is to map the n private inputs to n private outputs, where the outputs are a (random) functionality of the inputs. The

simplest case is the one in which $n = 2$, *i.e.*, secure two-party computation.

A standard communication model of secure multi-party computation assumes the existence of a private communication channel between each pair of parties. If $n > 2$, for simplicity, the standard model assumes that the communication is synchronous. Sometimes, we assume a broadcast channel in addition. It is clear that this additional broadcast channel can be simulated in the standard communication model using a Byzantine agreement protocol, provided that more than two thirds of the participants are honest.

With respect to the computational power of the participants, there are two popular models — bounded and unbounded. In the bounded model, each party is a probabilistic Turing Machine running in polynomial time; in the unbounded model, no restriction is placed on the computational power of any party. (Sometimes a mixed model is considered, where some parties are computationally unbounded, and others are bounded.)

Proof of security of multi-party computation is based on the idea of simulation. A protocol is considered secure if the joint view of all parties can be "simulated" in an ideal model, in which each participant sends her input to an additional trusted party and gets back the corresponding output. Note that simulation has different meanings in different computational models: In the computationally bounded model, it is sufficient that the two views be *computationally indistinguishable*; in the computationally unbounded model, it is required that the two views be identically distributed. Security in the former scenario is called "computational security," and in the latter scenario "information-theoretic security." A third type of simulation requires that the two views be statistically indistinguishable, which is more strict than computational indistinguishability but less strict than identical distribution. More details of these definitions will be provided as needed in Section 2.1.

Two adversarial models are often studied in secure multi-party computation: *semi-honest* and *malicious*. A semi-honest (*i.e.*, honest-but-curious) party follows the protocol faithfully but may attempt to compute information about the honest participants' inputs. In contrast, a malicious party is allowed to deviate arbitrarily from the protocol. In either case, it is implicitly assumed that all dishonest (semi-honest or malicious, respectively) parties collude. As pointed out in [59], we can conveniently suppose that all of them are controlled by one single adversary.

1.2.2 Existing Works in Secure Multi-party Computation

Completeness Theorems The problem of secure multi-party computation has been studied extensively. In particular, Yao [142] and Goldreich, et al. [60] presented very general results in the computationally bounded model, for two parties and multiple parties, respectively,

In [142], Yao showed that, if trapdoor permutations exist, then any two-party functionality can be securely computed. This is true even if either party is malicious. To obtain this result, Yao used Goldreich, Micali, and Wigderson's "compiler" [61] to reduce the problem with a malicious adversary to that with a semi-honest adversary.

In [60], Goldreich, Micali, and Wigderson further proved that, if trapdoor permutations exist, then any multi-party functionality can be securely computed, with respect to a semi-honest adversary. Combined with [61], this result essentially implies that any multi-party functionality can also be securely computed with respect to a malicious adversary. More precisely, secure multi-party computation can be achieved in two senses: In the first sense, the computation is either completed (with correct outputs) or *aborted* (if there is cheating), without revealing any extra information beyond the outputs (if any) to the adversary in either case; in the second sense, under the assumptions that *the majority of the participants are honest* and

that broadcast channel is available, the computation is always completed successfully and no information (other than the desired output) is revealed.

In the computationally unbounded model, a similar completeness theorem was proved by Ben-Or, Goldwasser, and Wigderson [16] and by Chaum, Crepeau, and Damgård [29], independently.

More Efficient Solutions The protocols constructed for the proofs of these completeness theorems are highly expensive. Consequently, more efficient solutions have been investigated.

Beaver, Micali, and Rogaway [12] considered round complexity in the computationally bounded model and showed that a constant number of rounds suffice. Franklin and Yung [47] and Gennaro, Rabin, and Rabin [53], independently, further reduced the communication complexity significantly. The most communication-efficient results in the semi-honest-adversary model were given by Hirt, Maurer, and Przydatek [70] in the computationally unbounded model and by Cramer, Damgård, and Nielson [35] in the computationally bounded model. Hirt and Maurer [69] presented a construction with the same communication complexity in the malicious-adversary model.

In many practical situations, (*e.g.*, in mobile-agent computation), it is necessary to finish the computational task after one round of interaction. Toward this end, Sander, Young, and Yung [126] proposed a solution to one-round, two-party secure function evaluation[1] for a log-depth circuit. Later, Cachin, Camenisch, Kilian, and Müller [25] presented a solution to this problem for any polynomial-size circuit.

Cramer and Damgård [34] studied an important class of secure computation — secure distributed linear algebra. Note that their solution differs from other

[1]Secure function evaluation is a variant of secure computation in which one party's output is null.

solutions mentioned above in that it is specifically designed for one specific class of computation. My proposed solutions are similar to theirs on this point (but not for the same class of computation).

1.2.3 Proposed Solutions vs. Existing Results

In this dissertation, I will propose solutions to various secure multi-party computation problems.

The first difference between my proposed solutions and the existing results mentioned above lies in efficiency. Each of my proposed solutions is specifically designed for a concrete problem and thus more efficient than standard general-purpose constructions. In particular, general-purpose protocols are often based on the evaluation of arithmetic circuits. However, in order to represent a conceptually simple functionality, typically we need a circuit of a very large size. Therefore, as pointed out by Goldreich [59], general-purpose protocols constructed in the proofs of completeness theorems should not be used for practical purposes. In contrast, my solutions are based on high-level operations and thus do not have this drawback.

Another difference lies in the communication model. The practical problems for which my solutions are designed often need non-standard communication models. For example, a private and synchronous communication channel may not be available between each pair of participants.

Yet another difference lies in the addition of incentive considerations. In particular, in Chapter 7, I attempt to consider the incentives of participants in the study of a multi-party computation problem in wireless ad hoc networks. As far as I know, this issue has not been well addressed in general.

1.3 Organization of Dissertation

The rest of this dissertation is organized as follows: Chapter 2 presents the basic definitions of secure multi-party computation and the techniques I frequently use. Chapter 3 presents joint work with Philippe Golle, Dan Boneh, Markus Jakobsson, and Ari Juels [66], which designs an efficient mix that is both private and verifiable. Chapter 4 presents joint work with James Aspnes, Joan Feigenbaum, and Aleksandr Yampolskiy [10], which discusses the possibilities and impossibilities of using entanglement to protect the integrity of data on a remote, untrusted server. Chapter 5 presents a set of privacy-preserving algorithms for distributed mining of association rules. Chapter 6 presents joint work with Yang Richard Yang [145], which introduces a new cryptographic primitive and applies it to the problem of secure mobile-agent computation. Chapter 7 presents joint work with Jiang Chen and Yang Richard Yang [144], which is an attempt to add incentive compatibility to secure multi-party computation. This work considers how to stimulate cooperation in mobile ad hoc networks. Chapter 8 concludes this dissertation.

Chapter 2

Technical Preliminaries

In this chapter, I first review the important definitions of secure multi-party computation. Then, I summarize the techniques I frequently use; these techniques will be the useful in various application, *e.g.*, mix networks, distributed data mining, and secure mobile-agent computation.

2.1 Definitions for Secure Computation

2.1.1 Preliminaries

For completeness, in this subsection, we review preliminary definitions from complexity theory that will be used in this dissertation.

Negligible and High Probabilities Denote by $\mathcal{N}^+$ the set of positive integers. Then we have the following standard definition of a negligible function.

Definition 1 *Suppose that* $\nu(\cdot) : \mathcal{N}^+ \to [0, 1]$ *is a function. We say* $\nu(\cdot)$ *is* negligible

if, for all positive polynomial $p(\cdot)$, there exists $n_0 \in \mathcal{N}^+$ such that, for any $n > n_0$,

$$\nu(n) < \frac{1}{p(n)}.$$

When we consider a computational problem, the probability of an event is a function of the length of input. We say that an event happens with *high probability* if the probability that it does not happen is negligible.

Computational Indistinguishability A *probability ensemble* is an (infinite) sequence of random variables with a bitstring index.

Definition 2 *Two probability ensembles $\{u\}_{i\in I}$ and $\{v\}_{i\in I}$ ($I \subseteq \{0,1\}^*$) are* computationally indistinguishable *if, for every polynomial-sized circuit family $\{C_n\}_{n\in\mathcal{N}^+}$,*

$$|Pr[C_n(u_i) = 1] - Pr[C_n(v_i) = 1]|$$

is a negligible function in the length of i. In such a case, we write

$$\{u_i\}_{i\in I} \stackrel{C}{\equiv} \{v_i\}_{i\in I}.$$

2.1.2 Secure Multi-party Computation

A secure n-party computation problem involves n parties, each having a private input x_i. The goal is to compute

$$f(x_1, \ldots, x_n) = (f_1(x_1, \ldots, x_n), \ldots, f_n(x_1, \ldots, x_n)),$$

where each $f_i(\cdot, \ldots, \cdot)$ $(i = 1, \ldots, n)$ is an n-ary function, such that the ith party obtains an output $f_i(\cdot, \ldots, \cdot)$. (Note that, by saying that each $f_i(\cdot, \ldots, \cdot)$ is a function,

we have actually required $f(x_1, \ldots, x_n)$ to be a *fixed*, rather than random, vector for each input vector $(x_1, \ldots, x_n)$. For the more general setting in which $f(\cdot, \ldots, \cdot)$ is a *random functionality*, please see [59].) For security, it is required that the privacy of any honest party's input is protected, in the sense that each dishonest party learns nothing except its own output. If there is any malicious party that may deviate from the protocol, it is also required that each honest party get a correct result whenever possible.

Security with Semi-honest Parties To precisely define the security in the case of semi-honest parties, we need first to define the *view* of each party during an execution of the protocol. During an execution of the protocol with input $(x_1, \ldots, x_n)$, the ith party's view, denoted by $view_i(x_1, \ldots, x_n)$, consists of this party's input x_i, all the coin flips of this party, and all the messages this party receives. Clearly, $view_i(x_1, \ldots, x_n)$ is a random variable. Therefore, $\{view_i(x_1, \ldots, x_n)\}_{(x_1,\ldots,x_n)\in\{0,1\}^*}$ is a probability ensemble.

Definition 3 *An n-party computation protocol for $f(\cdot, \ldots, \cdot)$ is secure with respect to semi-honest parties if, for each $i \in \{1, \ldots, n\}$, there exists a probabilistic polynomial algorithm M such that*

$$\{M(x_i, f_i(x_1, \ldots, x_n))\}_{x_1,\ldots,x_n\in\{0,1\}^*} \stackrel{C}{\equiv} \{view_i(x_1, \ldots, x_n)\}_{x_1,\ldots,x_n\in\{0,1\}^*}.$$

Security with fully malicious parties For the general definition of security with fully malicious parties, please refer to [59]. I am not including this definition here because I will not directly use it in this dissertation. In my work on mix networks (Chapter 3) and mobile agents (Chapter 6), I will consider malicious parties; however, security will be defined and proved for the specific applications therein, so that it

will be simpler and more understandable.

2.2 Frequently Used Techniques

2.2.1 ElGamal Encryption

ElGamal Encryption This is a probabilistic encryption scheme. Denote by $\mathcal{P}$ the set of primes. Define

$$\mathcal{Q} \stackrel{def}{=} \{q | p = 2q + 1 \in \mathcal{P}, q \in \mathcal{P}\}.$$

For $q \in \mathcal{Q}$ and $p = 2q + 1$, denote by G_q the quadratic-residue subgroup of Z_p^*. Consider a group family $\{G_q\}_{q \in \mathcal{Q}}$. Suppose that g is a generator of G_q. Let $k \in [0, |G_q| - 1]$ be a private key, and $K = g^k$ the corresponding public key. Suppose that $m \in G_q$ is a cleartext message. Then an encryption of m under the key pair (k, K) is

$$C = (mK^r, g^r),$$

where r is chosen uniformly at random from $[0, |G_q| - 1]$.

To decrypt a ciphertext $C = (C_1, C_2)$ using the private key k, one can compute

$$m = \frac{C_1}{C_2^k}.$$

To re-randomize a ciphertext $C = (C_1, C_2)$, one can compute

$$C' = (C_1 \cdot K^{r'}, C_2 \cdot g^{r'}),$$

where r' is chosen uniformly at random from $[0, |G_q| - 1]$.

Decisional Diffie-Hellman Assumption The security of ElGamal encryption is based on the *Decisional Diffie-Hellman* (DDH) assumption [18]. Denote by $Gen(G_q)$ the set of generators of G_q. For uniformly random $a, b, c \in [0, |G_q| - 1]$, the DDH assumption states that

$$\{(g^a, g^b, g^c)\}_{q \in \mathcal{Q}, g \in Gen(G_q)} \stackrel{C}{\equiv} \{(g^a, g^b, g^{ab})\}_{q \in \mathcal{Q}, g \in Gen(G_q)}.$$

It is well known that the ElGamal encryption is semantically secure in the sense of [63] if the DDH assumption holds.

2.2.2 Secret Sharing and Threshold Decryption

Feldman Verifiable Secret Sharing Let $k \in [0, q-1]$ be a secret. If we share k using the Feldman Verifiable Secret Sharing (VSS) among n parties with threshold t, then k_i, the ith share of k, is exactly the ith share of k using the well-known (n, t)-Shamir secret sharing. In addition, g^{k_i} is made public as the ith party's commitment to k_i. The advantage of the Feldman VSS is that the commitments allow us to do various computations related to the secret shares without compromising privacy.

One way to set up the Feldman VSS is to use a single trusted party. Another possible way to set up the Feldman VSS is to use a general-purpose protocol for secure multi-party computation (like those constructed for completeness theorems), which is less efficient. However, we need to set up VSS only once, and then we can reuse the set-up for multiple sessions. Therefore, both ways to set up the Felman VSS will be acceptable. (Although Pedersen's protocol [114] was proposed to set up VSS efficiently without a trusted party, it has been pointed out that there is a flaw in this protocol [52].)

Desmedt-Frankel Decryption If a private key k is shared among n parties with threshold t, Desmedt and Frankel [40] show that any quorum of t parties can jointly decrypt the ElGamal ciphertexts without explicitly reconstructing k. Let k_i be the share of the ith party, and $K_i = g^{k_i}$ the commitment to k_i. A quorum T of t parties can decrypt a ciphertext (C_1, C_2) as follows:

$$D_k(C_1, C_2) = \frac{C_1}{C_2^k} = \frac{C_1}{\prod_{i \in T}(C_2^{k_i}) \prod_{j \in T, j \neq i} \frac{-j}{i-j}}.$$

Observe that this equation requires each party $i \in T$ to raise C_2 to the k_i-th power. If necessary, party i may prove that it has honestly computed $S = C_2^{k_i}$ with the following proof of discrete-logarithm equality:

$$\log_{C_2} S = \log_g K_i (= k_i).$$

If only semi-honest adversaries are considered, then the proof is not necessary.

2.2.3 Keep-or-Randomize and Keep-or-Replace Operations

In this subsection, we consider two special operations. To define these two operations, we give two new notations.

The operator $\kappa_b()$ is parameterized by an identity b. If b holds, $\kappa_b()$ returns the input; otherwise it returns a random element chosen uniformly from G.

The operator $\iota_{b;w}()$ is similar to $\kappa_b()$; the only difference is that $\iota_{b;w}()$ returns $w \in G$, not a random element, if b does not hold.

Keep-or-Randomize Suppose that e, f are two random variables, whose ElGamal encryptions, E, F are public. Now I show a technique to compute an ElGamal encryption of $\kappa_{e=v}(f)$ ($v \in G_q$ is a constant) without knowing the private key or

decrypting E, F. Note that this technique is similar to the well-known technique of *selective disclosure* [7], but it keeps, rather than discloses, f if $e = v$ holds.

Suppose that $E = (E_1, E_2)$ and $F = (F_1, F_2)$. To achieve the above goal, any involved party can compute

$$((E_1/v)^r \cdot F_1, (E_2)^r \cdot F_2),$$

where r is a random element of $[0, |G_q| - 1]$. Then it re-randomizes this ciphertext. It is straightforward to see that the result is a re-randomization of F if $e = v$ and that it is a random encryption of a random cleartext if $e \neq v$.

Keep-or-Replace Again, suppose that e is a random variable whose ElGamal encryption E are public. I show a technique for party i to compute an ElGamal encryption of $\iota_{x_i=v;w}(e)$ ($v, w \in G_q$ are constants) without knowing the private key or decrypting E, where x_i is party i's private input.

To achieve the above goal, party i compares x_i and v, if they are equal, then it re-randomizes E; otherwise, it computes an encryption of w.

Chapter 3

An Efficient Mix

A mix network is a series of servers for anonymizing traffic. In this dissertation, I mainly consider decryption mix networks. In the setting of a decryption mix network, a set of input messages is fed to the first mix server in an encrypted form. Then each mix server processes the messages and forward them to the next server. Finally, the outputs of the last mix server are decrypted,[1] and the decryptions are supposed to the cleartexts of the inputs in a random order, such that the association between the inputs and the outputs is hidden. Obviously, the process of mixing is a special case of computation with untrusted parties. Privacy and verifiability will be the major security concerns.

In this chapter, I present joint work with Phillippe Golle, Dan Boneh, Markus Jakobsson, and Ari Juels [66]. Using some techniques from Chapter 2 and some specifically designed for this problem, we propose a new mix network that is optimized to produce a correct output very fast when all mix servers execute the mixing protocol correctly (the usual case). Our mix network only produces an output if no server cheats. However, in the rare case when one or several mix servers cheat, we

[1] In some decryption mix networks, (partial) decryption is also performed while the messages are forwarded.

convert the inputs to a format that allows "back-up" mixing. This back-up mixing can be implemented using any one of a wide array of already proposed (but slower) mix networks. When all goes well, our mix net is the fastest, both in real terms and asymptotically, of all those that offer standard guarantees of privacy and correctness. In practice, this benefit far outweighs the drawback of a comparatively complex procedure to recover from cheating. Our new mix is ideally suited to compute almost instantly the output of electronic elections, whence the name "exit-poll" mixing.

Note that our original design in [66] was found flawed by Wikström [141]. The mix presented in this chapter is an fixed version basd on what Wikström suggests in [141]. In particular, this fix adds to our original design a proof of knowledge of plaintext input, which guarantees that all inputs are independent and therefore prevents relation attacks [78].

3.1 Background and Motivation

The recently devised mix network constructions of Furukawa and Sako [50] and Neff [104] provide the full spectrum of security properties desirable in an election scheme. They achieve privacy, which is to say concealment of individual votes, and also robustness against Byzantine server failures. They additionally possess the property of *universal verifiability*, that is, the ability for any entity to verify the correct functioning of the mix, even in the face of an adversary that controls all servers and voters. Finally, the Furukawa-Sako and Neff mixes are substantially more efficient in terms of both computational and communications requirements than previously proposed mix networks with similar security properties.

Fast as they are, however, these mixes still remain cumbersome as tools for large-scale elections. Furukawa, *et al.* [49] report a running time of roughly six hours to

process a batch of 100,000 votes. In a federal election involving large precincts (conceivably millions of ballots in some states) a complete tally would thus require many hours. Premature media predictions of Gore's victory in Florida in the 2000 U.S. presidential election demonstrate the hunger of the electorate for timely information, and also the mischief that can be wrought in its absence. There is clearly a political and social need for faster tallying mechanisms than Furukawa-Sako and Neff alone can provide.

We describe here a mix network that is tailored for election systems, but with a substantial speedup over Furukawa-Sako and Neff. In settings like that described by Furukawa *et al.*, for example, we estimate that our construction is capable of yielding a six-to-eight times speedup. We achieve this improvement by taking an "optimistic" or "fast-track" [53] approach. In particular, we identify functionality in Furukawa-Sako and Neff that is not needed in the likely case that mix servers behave correctly and that most ballots are well formed. In the optimistic case, we show how to dispense with the costly property of robustness against Byzantine server failures. We also provide a form of universal verifiability that is somewhat weaker than the standard definition, but less costly, and adequate for nearly all types of elections, as we explain.

We refer to our proposal as an *exit-poll* mix network, by analogy with the "exit polls" used to provide fast predictions of election outcomes. If servers behave correctly, our exit-poll mix yields a correct election tally very rapidly. We expect this to be by far the most common case. If server cheating occurs, our mix identifies misbehaving servers. The privacy of all votes remains protected (given a majority of honest servers), but our mix does not produce an output. In such cases, our exit-poll scheme permits seamless fallback to a more heavyweight mix (like Furukawa-Sako or Neff) which can take over, complete the mixing and produce an output. Such heavyweight

mixes can also be employed to achieve supplemental, after-the-fact certification of an election tally achieved with our mix.

While our mix network is designed particularly for use in election schemes, we note that it can be employed in many of the other applications. Examples include anonymous e-mail [28] and bulletin boards, anonymous payment systems [83] as well as anonymized Web browsing [51].

The rest of this chapter is organized as follows. Related work is reviewed in Section 3.2. In Section 3.3, an ElGamal re-encryption mix networks is described. The high-level design of our new mix network is presented in Section 3.4, and a detailed description of the protocol is given in Section 3.5. In Section 3.6, the security properties of our mix network is analyzed. Section 3.7 summarizes this work on mix.

3.2 Related Work

Chaum proposed the first mix network, a decryption mix, in [28]. In Chaum's construction, users encrypt their inputs with the public-key of each mix server, starting with the last and ending with the first mix server in the net. Each mix server removes one layer of encryption, until the plaintexts are output by the last server. The weakness of this approach is that the mixing can not proceed if a single server is unavailable. To achieve robustness against server failures, Park, Itoh, and Kurosawa [113] introduced a new type of mix, re-encryption mix nets, in which the mixing and decryption phases are separated (see Section 3.3). The particular re-encryption mix of Park-Ito-Kurosawa was shown insecure in [117, 116], but was later fixed in [108].

The main difficulty of re-encryption mixes is to design computationally efficient ways for mix servers to *prove* that they mixed and re-encrypted their inputs correctly

in the mixing phase. The first techniques were based on costly general purpose cut-and-choose zero-knowledge proofs [124, 108, 1]. Millimix [81] and MIP-2 [2, 3] are based on more efficient zero-knowledge proofs specifically designed to prove that an output is a re-encryption of one of two inputs.

The most efficient schemes before our design that offer the full spectrum of security properties are those of Furukawa and Sako [50] and Neff [104]. The table in Figure 3.2 compares the real cost of mixing n items (in terms of the number of exponentiations required) with different mixing schemes (the numbers are taken from the respective papers). The column indicating the cost of proof and verification is in bold, because that is typically by far the most expensive step, and it is the step that we are optimizing. The cost of re-encryption is higher in our scheme than in others, but the difference pales in comparison with our savings in the proof and verification step. Furthermore, the re-encryption exponentiations can be pre-computed. The table also indicates whether each mixing scheme can take advantage of the speed-up techniques proposed in [79][2] for multiple exponentiations with respect to a fixed base. These techniques, based on addition chains, reduce the equivalent cost of one exponentiation to approximately 10 multiplications for reasonable sizes of batches (see [79] for more details). This amounts to a very significant speed-up. Our scheme is not only the fastest, but also the only one that can fully take advantage of addition chains in this sense.

An attractive alternative to mix networks is homomorphic encryption, in particular the Paillier scheme [111]. Election schemes based on homomorphic encryption require a good deal of computation for verification of correct ballot formation, but

[2]Other aspects of that proposal were later found flawed, and corrected, in [98]. The exposed vulnerabilities do not affect the soundness of the speed-up techniques.

[3]We note that these proposals have computational costs quadratic in the number of servers, due to the use of interactive proofs. However, if non-interactive proofs are employed – as in subsequent papers – this is brought down to a linear cost. The computational cost we use in the table assumes that this enhancement is performed.

Scheme	Re-encrypt	**Proof and verification**	Decrypt	Addition chain speed-up?
Cut and choose[3][124, 108]	$2n$	$642nm$	$(2+4m)n$	no
Pairwise permutation[2, 81]	$2n$	$7n \log n(2m-1)$	$(2+4m)n$	partially
Matrix representation[50]	$2n$	$18n(2m-1)$	$(2+4m)n$	partially
Polynomial scheme[104]	$2n$	$8n(2m-1)$	$(2+4m)n$	partially
Exit-poll mixing [this work]	$6n$	$6+12m$	$(5+10m)n$	yes

Figure 3.1: Optimistic cost per server (for a total of m servers) of mixing n items with different mix schemes, measured in number of exponentiations. Note that our proof and verification costs do not depend on n. The table also indicates whether addition chains can be used to pre-compute exponentiations. "Partially" indicates that addition chains can be used only in the mixing phase but not to prove correctness.

very little for tallying. In practice, therefore, they can be much faster than mix-based election schemes. An objection to homomorphic schemes has been their inability to accommodate write-in votes, an unavoidable requirement in the election systems of many jurisdictions. Kiayias and Yung [88] have devised a simple scheme that circumvents this difficulty. In brief, the idea is to permit each ballot to contain either a standard vote or a write-in vote, and to set aside write-in votes for separate processing via a mix network (in the unlikely case that this is needed).

It is our belief that mix networks will nonetheless remain an essential tool in electronic voting, as they still provide features that homomorphic schemes cannot. Vote-buying and coercion are serious threats in any election, but potentially much more problematic in Internet-based elections, given the anonymizing mechanisms available on the Internet and its reach across many jurisdictions. Schemes based on mix networks offer ways of minimizing these threats [71], while homomorphic schemes do not. A second advantage of mix networks is their flexibility with regard to key distribution. To distribute shares in the Paillier system without use of a trusted third party requires expensive joint RSA key-generation protocols (e.g., [19]), and distribution of a fresh RSA modulus for every election involving a different

distribution of trust. Mix-based schemes can be based on discrete-log cryptosystems, with simpler and more generalizable keying mechanisms. With this in mind, we propose a new mix network which offers a significant efficiency improvement over existing constructions.

3.3 An ElGamal Re-encryption Mix Network

In this section, we describe the basic operation of a re-encryption mix network based on the ElGamal cryptosystem. It will serve as a basis for our main construction described in Sections 3.4 and 3.5. The operation of a mix network can be divided into the following steps:

1. **Setup phase.** In the setup phase, the mix servers jointly generate the public and private keys of an ElGamal cryptosystem. The private key is shared in a (t, n)-Feldman VSS among all mix servers, while the public key is published.

2. **Submission of inputs.** All users submit their inputs to the mix encrypted with the public key generated in the setup phase.

3. **Mixing phase.** Each mix server in turn mixes and re-randomizes the batch of ciphertexts submitted to the mix.

4. **Decryption phase.** After the mixing is done, all output ciphertexts are decrypted by a quorum of mix servers, using the Desmedt-Frankel decryption.

Recall that ElGamal is a randomized encryption scheme that allows for re-randomization of ciphertexts. Given an ElGamal ciphertext (C_1, C_2), a mix server can efficiently compute a new ciphertext (C_1', C_2') that decrypts to the same plaintext as (C_1, C_1). To re-randomize a ciphertext, the mix server chooses a value $r \in Z_Q$

uniformly at random and computes $(C_1', C_2') = (C_1 K^r, C_2 g^r)$, where K is the public key. Given two ElGamal ciphertexts, it is infeasible to determine whether one is a re-randomization of the other without knowledge of either the private decryption key k or the re-randomization factor r. A mix server can use this property to hide the correspondence between its input and output ciphertexts: the input ciphertexts are first re-randomized, then output in a random order.

However, a mix server who knows the re-randomization factor r can efficiently convince a verifier that (C_1', C_2') is a re-randomization of (C_1, C_2) without revealing r. The proof of re-randomization consists simply of proving that

$$\log_g(C_2'/C_2) = \log_K(C_1'/C_1),$$

which trivially implies that there exists r such that $(C_1', C_2') = (C_1 K^r, C_2 g^r)$. To prove the foregoing discrete logarithm equality, we may use, for example, Schnorr signatures [129] (as suggested by Jakobsson [78]) or a non-interactive version [46] of the Chaum-Pedersen protocol [30]. This proof of re-randomization will serve as the basis for a proof that allows a mix server to prove that it mixed its inputs correctly (observe that in the real proof of correctness, a mix server must not reveal which output is a re-randomization of which input, so the proof outlined above will not work *as is*.)

3.4 Mix Net Design

Our new mix net mixes ciphertexts like an ElGamal re-encryption mix. The novelty lies first in a highly efficient method for proving that the mixing was done correctly, and second in a method for falling back on a more heavyweight mix if cheating by

a server is detected. We start with a high-level description of these two building blocks.

Each input ciphertext submitted to our mix net is required to be the encryption of a plaintext that includes a cryptographic checksum. To verify that a mix server operated correctly, we ask for a proof that the product of the plaintexts corresponding to the input ciphertexts equals the product of the plaintexts corresponding to the output ciphertexts. As we shall show, such proofs can be produced and verified highly efficiently without knowledge of the plaintexts. We call this proof a *proof-of-product (POP) with checksum.*

This proof however does not detect all types of cheating. Rather, it guarantees that if the mix server did not mix correctly, it had to introduce in the output at least one new ciphertext that corresponds to a plaintext with an invalid checksum. When outputs are decrypted, invalid checksums are traced to one of two sources: either an input that was originally submitted to the mix network with an invalid checksum, or a cheating mix server. The difficulty of this approach lies in the fact that, because invalid checksums can only be traced at decryption time, cheating may not be detected until after the harm is done. In effect, a cheating server may be able to match inputs to outputs before cheating gets detected in the verification step. If we were to use this mix just like that, nothing could be done after a server has cheated to restore the privacy of those users whose inputs have already been traced. In particular, a second round of mixing wouldn't help.

To address this difficulty, we introduce the second main contribution of this work, which may be of interest on its own. Our approach is to encrypt users' inputs twice (a technique we call *double enveloping*). In the verification step outlined above, the output ciphertexts are first decrypted only once. If the verification succeeds and no servers are found to have cheated, the output ciphertexts are decrypted one more time

and yield the plaintext. If, on the other hand, one or several servers are found to have cheated, the output ciphertexts are not decrypted further. Instead, they become the input to a different (slower) mix network such as Neff [104] and are mixed a second time before being finally decrypted. This second round of mixing ensures that the privacy of users can not be compromised. A cheating server in the first round of mixing may learn at most the relationship between a double-encrypted ciphertext and a single-encrypted ciphertext, which does not help to find the corresponding plaintext after the second round of mixing.

In the rest of this section, we describe these two building blocks in greater detail.

3.4.1 Proof of Product with Checksum.

Consider a mix server who receives as inputs n ElGamal ciphertexts $(C_1^{(i)}, C_2^{(i)})$, and outputs a permuted re-randomization of these, namely a permutation of the list of $(C_1'^{(i)}, C_2'^{(i)}) = (C_1^{(i)} K^{r_i}, C_2^{(i)} g^{r_i})$. Our key idea is to let the mix server prove that its operations are *product preserving*, i.e. that the product of the plaintexts corresponding to the input ciphertexts $(C_1^{(i)}, C_2^{(i)})$ equals the product of the plaintexts corresponding to the output ciphertexts $(C_1'^{(i)}, C_2'^{(i)})$.

Using the homomorphic property of ElGamal encryption, any verifier can compute an ElGamal encryption (C_1, C_2) of $\prod m_i$, and an ElGamal encryption (C_1', C_2') of $\prod m_i'$, where m_i (resp. m_i') is the plaintext corresponding to $(C_1^{(i)}, C_2^{(i)})$ (resp., $(C_1'^{(i)}, C_2'^{(i)})$). To prove that its operations are *product preserving*, the mix server only needs to prove that

$$\log_K(C_1'/C_1) = log_g(C_2'/C_2).$$

As we saw in Section 3.3, this implies that $\prod m_i = \prod m_i'$.

The Need for a Checksum The product equality $\prod m_i = \prod m'_i$ clearly does not imply that the sets $\{m_i\}_{i=1}^n$ and $\{m'_i\}_{i=1}^n$ are equal. In other words, the property of being product-preserving does not by itself guarantee that a mix net operates correctly. Our approach is to restrict the plaintexts m_i (and therefore also m'_i) to a particular format, in such a way that it becomes infeasible for a dishonest mix server to find a set $\{m'_i\} \neq \{m_i\}$ such that $\prod m_i = \prod m'_i$ *and* all the elements m'_i are of the required format. We propose to define this special format by adding a cryptographic checksum to the plaintext, drawing on the techniques of Jakobsson-Juels [82]. This is done as follows.

Users format their inputs to the mix net as an ElGamal encryption of a plaintext m and an ElGamal encryption of $h(m)$, where h is a cryptographic hash function (in security analysis, we model h has a random oracle [15]):

$$\Big(E_K(m, r), E_K(h(m), r')\Big)$$

Each input to the mix now consists of a *pair* of ElGamal ciphertexts. The mix re-randomizes separately each of the two ElGamal ciphertexts in every pair, then outputs all the pairs in a random order. The mix must then prove that the products of the plaintexts corresponding to the first element in the pair are the same in the input and the output ($\prod m_i = \prod m'_i$) and also that the products of the plaintexts corresponding to the second element in the pair are the same in the input and the output ($\prod h(m_i) = \prod h(m'_i)$). As we shall prove in Section 3.6, these two proofs together guarantee the set equality $\{m_i\} = \{m'_i\}$.

3.4.2 Double Enveloping

As we have already pointed out, a mix whose correctness was enforced only by a proof-of-product with redundancy may not detect server cheating until after the harm is done. To illustrate how users' privacy may be compromised even if all cheating servers are disqualified, we offer the following example. Assume that the first mix server is corrupt and that the input submitted by user i is $(E_K(m_i, r_i), E_K(h(m_i), r'_i))$. The corrupt first server can replace the input of user 1 by

$$(E_K(m_1, r_1)E_K(m_2, r_2), E_K(h(m_1), r'_1)E_K(h(m_2), r'_2)),$$

(recall that we define the product of vectors as a vector of products of the corresponding components), and replace the input of user 2 by $(1, 1, 1, 1)$. Such cheating will only be detected after the decryption phase. Even if the cheating server were to be disqualified and the mixing protocol restarted, the cheating server would still be able to distinguish the plaintexts submitted by users 1 and 2 from other users' plaintexts, by comparing the output of the restarted protocol with that of the first execution.

To defend against this attack, we add a second layer of encryption to the plaintext m of a user. A user whose plaintext input is m is required to submit the following triple of ciphertexts to the mix:

$$(E_K(C_1, r), E_K(C_2, r'), E_K(h(C_1, C_2), r'')),$$

where

$$(C_1, C_2) = E(m, \hat{r}) \stackrel{\triangle}{=} (m(K')^{\hat{r}}, K^{\hat{r}}).$$

(In the above inner-layer encryption, we use a new public key K' and replace the

canonical generator g with the outer-layer public key K. This technique, suggested by Wilkström [141], will allow the user to prove his knowledge of the plaintext and thus avoid a flaw of our original design in [66]. The proof will be detailed in Section 3.5.) If cheating is caused by a corrupted server, we can re-randomize all the inner-layer encryptions and their order, with a standard ElGamal-based re-randomization mix net, before they are finally decrypted to plaintexts. Although the adversary might be able to link some inner-layer encryptions to the input ciphertexts, he cannot link the final output plaintexts to them.

3.5 Exit-Poll Mix Net

Assumptions As in Chapter 2, we assume that there exists a bulletin board, which is accessible to the public, and is authenticated, tamper-proof, and resistant to denial-of-service attacks. All messages and proofs are posted on this bulletin board.

Setup The mix servers jointly generate parameters (p, q, g, k, K) for an ElGamal cryptosystem E, and an additional pair of keys (k', K') (which uses K, not g, as generator). The public parameters and the additional public key K' are made public, while the private keys k, k' are shared among the mix servers in a (t, n)-Feldman VSS scheme. Users are required to submit their input m_i to the mix net formatted as follows:

1. The user encrypts the input m_i to produce $E_{K'}(m_i) = (C_1^{(i)}, C_2^{(i)}) = (m_i(K')^{\hat{r}}, K^{\hat{r}})$.

2. The user computes $H^{(i)} = h(E_{K'}(m_i))$. As explained earlier, we model h as a random oracle in the proof of security. In practice, a publicly available hash function such as MD5 [120] or SHA-1 [107] should be used.

3. The user submits the triple $E_K(C_1^{(i)}), E_K(C_2^{(i)}), E_K(H^{(i)})$. The mix servers check that every component belongs to G_q, and that this input has not already been submitted. If any component is not in G_q, the user is disqualified and the triple is discarded. If the same input has already been submitted by another user, the duplicate submission is discarded.

4. The user proves his knowledge of the plaintext. (In the original design in [66], we only require the user to prove his knowledge of $C_1^{(i)}, C_2^{(i)}, H^{(i)}$, but this was found flawed by Wikström [141]. The following description is based on what Wikström suggests in [141].) To achieve this goal, the user only needs to prove that he knows all the random elements used for encrypting the plaintext. Recall that

$$\begin{aligned}(E_K(C_1^{(i)}), E_K(C_2^{(i)}), E_K(H^{(i)})) &= ((C_1^{(i)}K^r, g^r), (C_2^{(i)}K^{r'}, g^{r'}), (H^{(i)}K^{r''}, g^{r''})) \\ &= ((m_i(K')^{\hat{r}}K^r, g^r), (K^{\hat{r}+r'}, g^{r'}), \\ &\quad (h(m_i(K')^{\hat{r}}, K^{\hat{r}})K^{r''}, g^{r''})).\end{aligned}$$

   It suffices for the user to prove knowledge of r, r', r'' and $\hat{r} + r'$. Any user who fails to give the proof is disqualified, and the corresponding input is discarded.

5. We note that dishonest users may submit inputs that are not properly formatted, in the sense that the equality $H^{(i)} = h(E_{K'}(m_i))$ does not hold. We stress that such improperly formatted inputs can *not* force our mix net to default to the slower back-up mixing. The only event that can trigger a default to the back-up mixing is cheating by one of the mix servers.

First stage: re-randomization and mixing This step proceeds as in all re-randomization mix nets based on ElGamal. One by one, the mix servers re-randomize

all the inputs and their order. (Note that the components of triples are not separated from each other during the re-randomization.) In addition, each mix net must give a proof that the product of the plaintexts of all its inputs equals the product of the plaintexts of all its outputs.

1. Each mix server reads from the bulletin board the list of triples corresponding to re-encryptions of $E_K(C_1^{(i)}), E_K(C_2^{(i)}), E_K(H^{(i)})$ output by the previous mix server: $\{(a_i \cdot K^{r_i}, g^{r_i}), (b_i \cdot K^{s_i}, g^{s_i}), (c_i \cdot K^{t_i}, g^{t_i})\}_{i=1}^{N}$. (Note that even if some servers have cheated, the ciphertexts can still be formatted like that, provided that every component belongs to G_q.)

2. The mix server re-randomizes the order of these triples according to a secret and random permutation. Note that it is the order of triples that is re-randomized, and that the three components $E_K(C_1^{(i)})$, $E_K(C_2^{(i)})$, and $E_K(H^{(i)})$ that make up each triple remain in order.

3. The mix server then re-randomizes each component of each triple independently, and outputs the results: $\{(a'_i \cdot K^{r'_i}, g^{r'_i}), (b'_i \cdot K^{s'_i}, g^{s'_i}), (c'_i \cdot K^{t'_i}, g^{t'_i})\}_{i=1}^{N}$.

4. The mix server proves that $\prod a_i = \prod a'_i$ and $\prod b_i = \prod b'_i$ and $\prod c_i = \prod c'_i$.

Second stage: decryption of the inputs

1. A quorum of mix servers jointly decrypt each triple of ciphertexts to produce the values $C_1^{(i)}$, $C_2^{(i)}$ and $H^{(i)}$, using the Desmedt-Frankel decryption technique.

2. All triples for which $H^{(i)} = h(C_1^{(i)}, C_2^{(i)})$ are called *valid*.

3. Invalid triples are investigated according to the procedure described below. If the investigation proves that all invalid triples are *benign* (only users cheated),

we proceed to Step 4. Otherwise, the decryption is aborted, and we continue with the back-up decryption.

4. A quorum of mix servers jointly decrypts the ciphertexts $(C_1^{(i)}, C_2^{(i)})$ for all valid triples. This successfully concludes the mixing. The final output is defined as the set of plaintexts corresponding to valid triples.

Special step: investigation of invalid triples The investigation proceeds as follows. The mix servers must reveal the path of each invalid triple through the various permutations. For each invalid triple, starting from the last server, each server reveals which of its inputs corresponds to this triple, and how it re-randomized this triple. The cost of checking the path of an invalid triple is one exponentiation per mix server (the same cost as that incurred to run one input through the mix net). One of two things may happen:

- **Benign case (only users cheated):** if the mix servers successfully produce all such paths, the invalid triples are known to have been submitted by users. The decryption is resumed after the incorrect elements have been removed.
- **Serious case (one or more servers cheated):** if one or more mix servers fail to recreate the paths of invalid triples, these mix servers are accused of cheating and replaced, and our mix terminates without producing an output. In this case, the inputs are handed over to the back-up mixing procedure described next.

Back-up mixing The *outer-layer* encryption of the inputs posted to the mix net is decrypted by a quorum of mix servers. The resulting set of *inner-layer* ciphertexts becomes the input to a standard re-encryption mix net based on ElGamal (using,

for example, Neff's scheme described in [104]). At the end of this second mixing, the ciphertexts are finally decrypted to plaintexts, which concludes the mixing.

3.6 Analysis of Efficiency and Security

Efficiency We start with a brief discussion of the efficiency of our scheme. The costs are as follows for a batch consisting of n inputs:

- Re-encryption and mixing: **linear** number of modular exponentiations ($6n$).
- Proof of correct mixing: **constant** number of modular exponentiations (but number of modular multiplications linear in n).
- Verification: **constant** number of modular exponentiations per server (but number of modular multiplications linear in n). The cost is also linear in the number of servers.
- Decryption: **linear** number of modular exponentiations ($(5 + 10m)n$ for m servers).

This makes our mix twice as fast as the fastest mix network before this work [104]. Furthermore, in our mix, the costs are incurred mostly in the re-encryption and decryption phases, which is similar to the Flash mix [79]. This is important because these two phases (unlike the proof phase) can benefit from the significant speed-up techniques developed by Jakobsson [79]. Using addition chains, we estimate that the cost of one exponentiation is roughly equivalent to 10 multiplications, with reasonably sized batches.

Correctness We now turn to the guarantees of correctness offered by our mix network.

Claim 4 *(Correctness) If all parties follow the protocol, the output of the mix net is a permuted decryption of the input.*

Because the set of plaintexts is preserved in re-randomizations, this follows immediately from the correctness of decryption.

Verifiability The verifiability of our mix net is a restricted form of universal verifiability in the sense that only the operation of the mix net on *valid* inputs (*i.e.*, the inputs that are well-formed according to our protocol) are universally verifiable. We call this restricted form of verifiability "public verifiability".

Definition 5 *(Public Verifiability) A mix net is* publicly verifiable *if there exists a polynomially bounded verifier that takes as input the transcript of the mixing posted on the bulletin board, outputs "valid" if the set of valid outputs is a permuted decryption of all valid inputs, and otherwise outputs "invalid" with high probability. Note that, to prove public verifiability, we consider an adversary that can control all mix servers and all users.*

Claim 6 *Our mix net is publicly verifiable if the discrete logarithm problem is hard in G_q.*

Proof: The proof proceeds by contradiction. We assume that one or several mix servers cheat during the execution of a mixing protocol, yet manage to produce a transcript that fools an outside verifier into believing that the mixing was done correctly. We show how to use these cheating mix servers to compute discrete logarithms in the group G_q. Our proof is based on the following lemma:

Lemma 7 *Let a and b be two elements of G_q. For random values r_i and s_i, we compute the following group elements: $h_i = a^{r_i} b^{s_i}$. Let S be a non-empty subset of*

the indices i's such that $\prod_{i \in S} h_i = 1$. With high probability, the knowledge of this set S allows us to compute $\log_a b$.

The proof of the lemma is trivial: if $\sum_{i \in S} r_i \neq \sum_{i \in S} s_i$, which happens with high probability, then $\log_a b = -(\sum_{i \in S} r_i)/(\sum_{i \in S} s_i)$.

Now let us order the mix servers by the order in which they mix the inputs, and consider the first cheating mix server. We denote the inputs given to this first cheating mix server as:

$$(E_K(C_1^{(1)}, r_1), E_K(C_2^{(1)}, r_1'), E_K(H^{(1)}, r_1'')), \ldots, (E_K(C_1^{(N)}, r_N), E_K(C_2^{(N)}, r_N'), E_K(H^{(N)}, r_N'')),$$

and its output as

$$(E_K(\overline{C}_1^{(1)}, r_1), E_K(\overline{C}_2^{(1)}, r_1'), E_K(\overline{H}^{(1)}, r_1'')), \ldots, (E_K(\overline{C}_1^{(N)}, r_N), E_K(\overline{C}_2^{(N)}, r_N'), E_K(\overline{H}^{(N)}, r_N'')),$$

For cheating to escape detection, the equation

$$\prod_i H^{(i)} = \prod_i \overline{H}^{(i)} \tag{3.1}$$

must hold, *and* in addition, we must have $\overline{H}^{(i)} = h(\overline{C}_1^{(i)}, \overline{C}_2^{(i)})$ for all i. Furthermore, because all servers before the first dishonest server are honest, the checksums in the input are all correct, and therefore $H^{(i)} = h(C_1^{(i)}, C_2^{(i)})$. Equation 3.1 can therefore be rewritten as:

$$\prod_i h(C_1^{(i)}, C_2^{(i)}) = \prod_i h(\overline{C}_1^{(i)}, \overline{C}_2^{(i)}). \tag{3.2}$$

Now recall that we model the hash function h as a random oracle. Each time a mix server queries h on a new input, we choose random values r_i and s_i and return $a^{r_i} b^{s_i}$ (we answer queries on inputs that have already been queried consistently). Because

the mix server cheated, Equation 3.2 gives us a non-trivial product relationship of the type that allows us to compute discrete logarithms in the group G_q according to Lemma 7, and this concludes the proof.

Privacy The above presented version of our mix network, based on the suggested fix in [141], offers the same guarantee of privacy as all mix networks based on ElGamal re-encryptions, e.g., [104].

3.7 Summary of the Work on Mix Network

We constructed a verifiable mix network that is extremely fast in case none of the mix servers cheat. This enables election officials to quickly announce the results in the common case when all mix servers honestly follow the mixing protocol. In case one or more of the mix servers cheat, our system detects the cheating server or servers and then redoes the mixing using one of the standard (slower) mix systems [104]. We emphasize that server cheating cannot compromise user privacy; it just causes the mixing process to run slower.

This work on mix uses some of the techniques I show in Chapter 2. It also uses additional techniques like internal checksum and double enveloping. Public verifiability is a security property against malicious adversaries, specifically defined for mix networks. Interested readers can compare it with the general definition of security against malicious adversaries in secure multi-party computation [59].

Chapter 4

Data Entanglement: Secure Storage with Untrusted Server

The previous chapter studies anonymization with an efficient mix network. In this chapter, I consider a different scenario: building secure storage with an untrusted server. To achieve this goal, one possible way is to *entangle* many users' data together, so that corrupting one user's data will lead to corruption of all other users'. Obviously, the process of entangling is also a special case of computation with untrusted parties. The major security concern here is, however, integrity. In joint work with James Aspnes, Joan Feigenbaum, and Aleksandr Yampolskiy [10], I study this methodology.

Using data entanglement in storage systems was first proposed in Dagster [136] and Tangler [96]. These systems split data into blocks in such a way that a single block becomes a part of several documents; these documents are said to be *entangled*. Dagster and Tangler use entanglement in conjunction with other techniques to deter a censor from tampering with unpopular data. In this work, however, we focus on entanglement only.

We argue that while Dagster and Tangler achieve their stated goals, they do not achieve ours. In particular, we prove that deleting a typical document in Dagster affects, on average, only a constant number of other documents; in Tangler, it affects virtually no other documents. This motivates us to propose two stronger notions of entanglement, called *dependency* and *all-or-nothing integrity*, which bind the users' data so that it is hard to delete or modify the data of any one user without damaging the data of all users. We study these notions in several submodels, which vary in whether they permit arbitrary tampering with the common data store or only destructive tampering, and in whether they restrict all users to a standard recovery algorithm or let some users adopt a non-standard algorithm supplied by the adversary. In each of these models, we not only provide mechanisms for limiting the damage done by the adversary, but also argue, under reasonable cryptographic assumptions, that no stronger mechanisms are possible.

4.1 Background and Motivation

Suppose that I provide you with remote storage for your most valuable information. I may advertise various desirable properties of my service: underground disk farms protected from nuclear attack, daily backups to chiseled granite monuments, replication to thousands of sites scattered across the globe. But what assurance do you have that I will not maliciously delete your data as soon as your subscription check clears?

To convince you that you will not lose your data at my random whim, I might offer stronger technical guarantees. Two storage systems proposed linking your data to the data of other users: Dagster [136] and Tangler [96]. The intuition behind these systems is that data are partitioned into blocks in a way that every block can be

used to reconstruct several documents. New documents are represented using some number of existing blocks, chosen randomly from the pool, combined with new blocks created using exclusive-or (Dagster) or 3-out-of-4 secret sharing [130] (Tangler). Two documents that share a server block are said to be *entangled*.

Entangling new documents with old ones provides an incentive to retain blocks, as the loss of a particular block might render many important documents inaccessible. Dagster and Tangler use entanglement as one of many mechanisms to discourage negligent or malicious destruction of data; others involve disguising the ownership and contents of documents and (in Tangler) storing documents redundantly. This work is motivated in part by the question of whether these additional mechanisms are necessary, or whether entanglement by itself can effectively deter malicious censorship.

We begin by analyzing the use of entanglement in Dagster and Tangler in Section 4.2. We argue that the notion of entanglement provided by Dagster and Tangler is not by itself sufficiently strong to discourage censorship by punishing data loss, as not enough documents get deleted on average if an adversary destroys a block for some targeted document. In particular, we show in Subsection 4.2.3 that destroying a typical document in Dagster requires destroying only a constant number of additional documents on average, even if the adversary is restricted to the very limited attack of deleting a single block chosen uniformly at random from the blocks that make up the document. The situation with Tangler is worse: deleting two blocks from a particular document destroys the document without destroying any others (which lose at most one block each) in the typical case.

Our objective in this work is to examine the possibility of obtaining stronger notions of entanglement, in which the fates of specific documents are directly tied together. These stronger notions might be enough by themselves to deter censor-

ship, in that destroying a particular document, even if done by a very sophisticated adversary, could require destroying most or all of the other documents in the system. A system that provides such tight binding between documents gives a weak form of censorship resistance; though we cannot guarantee that no documents will be lost, we can guarantee that no documents will be lost unless the adversary burns down the library. Under the assumption that the adversary can destroy data at will, this may be the best guarantee we can hope to offer.

In Section 4.3, we define our model for a document-storage system in which the adversary is allowed to modify the common data store after all documents have been stored. Such modifications may include the block-deletion attacks that Dagster and Tangler are designed to resist, but they may also include more sophisticated attacks such as replacing parts of the store with modified data, or superencrypting all or part of the store.

In addition to modifying the data store, in some variants of the model the adversary is permitted to carry out what we call an *upgrade attack* (see Subsection 4.3.2), in which the adversary offers all interested users the choice of adopting a new algorithm to recover their data from the common store if they find that the old one no longer works. Allowing such upgrade attacks is motivated by the observation that a selfish user will jump at the chance to get his data back (especially if he has the ability to distinguish genuine from false data) if the alternative is losing the data. Upgrade attacks also exclude dependency mechanisms that rely on excessive fastidiousness in the recovery algorithm, as one might see in a recovery algorithm that politely declines to return its user's data if it detects that some other user's data has been lost.

In Section 4.4, we propose two stronger notions of entanglement in the context of our model: *document dependency* and *all-or-nothing integrity*. If a document

of user A depends on a document of user B, then A can recover her document only if B can. Unlike entanglement, document dependency is an externally-visible property of a system; it does not require knowing how the system stores its data, but only observing which documents can still be recovered after an attack. All-or-nothing integrity is the ultimate form of document dependency, in which every user's document depends on every other user's, so that either every user gets his data back or no user does. Our stronger notions imply Dagster's and Tangler's notion of entanglement but also ensure that an adversary cannot delete a document without much more serious costs.

The main part of this work examines what security guarantees can be provided depending on the assumptions made in the model. In Section 4.5, we consider how the possibility or impossibility of providing document dependency is determined by the choice of permitted adversary attacks. Subsection 4.5.1 shows that detecting tampering using a MAC suffices for obtaining all-or-nothing integrity if all users use a standard "polite" recovery algorithm. Subsection 4.5.2 shows that all-or-nothing integrity can no longer be achieved for an unrestricted adversary if we allow upgrade attacks. In Subsection 4.5.3, we show how to obtain a weaker guarantee that we call *symmetric recovery* even with the most powerful adversary; here, each document is equally likely to be destroyed by any attack on the data store. This approach models systems that attempt to prevent censorship by disguising where documents are stored. In Subsection 4.5.4, we show that it is possible to achieve all-or-nothing integrity despite upgrade attacks if the adversary can only modify the common data store in a way that destroys entropy, a generalization of the block-deleting attacks considered in Dagster and Tangler.

Finally, in Section 4.6, we discuss the strengths and limitations of our approach, and offer suggestions for future work on this topic.

4.1.1 Related Work

The problem of building reliable storage using untrusted servers has been studied for a long time. We distinguish between three basic approaches: replication, tamper detection, and entanglement. We also discuss the all-or-nothing transform, an unkeyed cryptographic tool that provides guarantees similar to all-or-nothing integrity.

Replication. Anderson, in his seminal paper describing an "Eternity Service" [9], proposed building a network of tamper-resistant servers spread around the world. Stored documents are redundantly replicated across the network, thereby making them *censorship-resistant* — difficult to delete without the cooperation of all servers. Many subsequent storage systems [27, 58, 99, 32, 139] have since implemented variations on this idea.

These systems solve a problem that is different from, and complementary to, the one considered in our work.

Our definition of all-or-nothing integrity applies just as well to distributed storage mechanisms as to centralized ones and reflects the concerns of users who care more about whether they will get their data back than how. Thus, even in a system that promises data availability through replication, all-or-nothing integrity provides additional assurance to the users by guaranteeing that any failure to fulfill this promise will carry a very high cost.

Tamper detection. A second approach to providing secure storage is based on detecting invalid modifications of the stored data. Two common tools used for tamper detection are digital signatures and Merkle hash trees [97]. Typical examples of such systems include NASD [56], S4 [135], SFSRO [48], SiRiUS [57], SUNDR [94, 95], and TDB [92].

The guarantee provided by these systems is rather weak. A user whose data is lost is likely to notice with or without being notified by the system. However, as we show in Section 4.5.1, tamper detection can be leveraged in to give all-or-nothing integrity if all users run standard recovery algorithms by the simple expedient of having all users politely refuse to recover their data if the store is tampered with. That some users might insist on recovering their uncorrupted data anyway points out a fundamental limitation of both the model of standard recovery algorithms and the approach of tamper detection.

Entanglement. To prevent impolite users from recovering their own data even if other users' data have been lost, two storage systems have been proposed that create dependencies between blocks of data belonging to different users: Dagster [136] and Tangler [96]. Because of their close connection to our work, we discuss these systems in Section 4.2.

All-or-nothing transform. Motivated by security problems in block ciphers, Rivest [121] proposed a cryptographic primitive called *all-or-nothing transform* (AONT). An all-or-nothing transform is an invertible transformation, which is similar to our notion of all-or-nothing integrity in the sense that either all bits of the preimage can be recovered (if the image is available) or none can be (if l bits of the image are lost) (see [26, 134]). However, it is also radically different because it does not involve multiple users who possess individual keys. Moreover, AONT does not consider the possibility that the image may be corrupted in other ways than some bits being deleted, such as the adversary superencrypting the entire data store.

4.2 Dagster and Tangler

We review how Dagster [136] and Tangler [96] work in Subsections 4.2.1 and 4.2.2, respectively. We describe these systems at the block level and omit details of how they break a document into blocks and assemble blocks back into a document. In Subsection 4.2.3, we analyze the intuitive notion of entanglement provided by these systems, pointing out some of this notion's shortcomings if it is the only mechanism provided to deter censorship.

4.2.1 Dagster

The Dagster storage system may run on a single server or on a P2P overlay network. Each document in Dagster consists of $c+1$ server blocks: c blocks of older documents and one new block, which is an exclusive-or of previous blocks with the encrypted document. The storage protocol proceeds as follows:

Initialization Upon startup, the server creates approximately 1000 random server blocks and adds them to the system.

Entanglement To publish document d_i, user i generates a random key k_i. He then chooses c random server blocks $C_{i_1}, \ldots, C_{i_c}$ and computes a new block

$$C_i = \mathcal{E}_{k_i}(d_i) \oplus \bigoplus_{j=1\ldots c} C_{i_j},$$

where $\mathcal{E}$ is a plaintext-aware encryption function[1] and $\oplus$ is bitwise exclusive-or. The user releases instructions on recovering d_i, which come in the form of a *Dagster Resource Locator* (DRL), a list of hashes of blocks needed to

[1] A plaintext-aware encryption function is one for which it is computationally infeasible to generate a valid ciphertext without knowing the corresponding plaintext. See [13] for a formal definition.

reconstruct d_i:

$$\left(k_i, \pi\left[h(C_i), h(C_{i_1}), H(C_{i_2}), \ldots, h(C_{i_c})\right]\right).$$

Here $h(\cdot)$ is a cryptographic hash function and π is a random permutation.

Recovery To recover d_i, the user asks the server for blocks with hashes listed in the DRL of d_i. If the hashes of the blocks returned by the server match the ones in the DRL, the user computes:

$$\mathcal{D}_{k_i}\left(C_i \oplus \bigoplus_{j=1\ldots c} C_{i_j}\right),$$

where $\mathcal{D}$ represents a decryption function. Otherwise, the user exits.

4.2.2 Tangler

The Tangler storage system derives its name from the use of $(4, 3)$-Shamir secret sharing [130] to entangle the data. Each document is represented by four server blocks, any three of which are sufficient to reconstruct the original document. The blocks are replicated across a subset of Tangler servers. A data structure similar to Dagster Resource Locator, called an *inode*, is used to record the hashes of the blocks needed to reconstruct the document. Here is the storage protocol:

Initialization As in Dagster, the server is jump-started with a bunch of random blocks.

Entanglement Each server block in Tangler is a pair (x, y), where x is a random element of $GF(2^{16})$ and y is the value of a polynomial at x. The polynomial is uniquely determined by any three blocks; it is constructed in a way that

evaluating it at zero yields the actual data. To publish document d_i, user i downloads two random server blocks, $C_{i_1} = (x_1, y_1)$ and $C_{i_2} = (x_2, y_2)$, and interpolates them together with $(0, d_i)$ to form a quadratic polynomial $p(\cdot)$. Evaluating $p(\cdot)$ at two different nonzero integers yields new server blocks C'_{i_1} and C'_{i_2}. The user uploads the blocks and records the hashes of the blocks comprising d_i (*i.e.*, $C_{i_1}, C_{i_2}, C'_{i_1}, C'_{i_2}$) in d_i's inode.

Recovery To recover his document, user i sends a request for blocks listed in d_i's inode to a subset of Tangler servers. Upon receiving three of d_i's blocks, the user can reconstruct $p(\cdot)$ and compute $d_i = p(0)$.

4.2.3 Analysis of Entanglement

Let us take a snapshot of the contents of a Dagster or Tangler server. The server contains a set of blocks $\{C_1, \ldots, C_m\}$ comprising documents $\{d_1, \ldots, d_n\}$ of a group of users. (Here $m, n \in \mathcal{N}^+$ and $m \geq n$.)

Data are partitioned in a way that each block becomes a part of several documents. We can depict this documents-blocks relationship using an *entanglement graph* (see Figure 4.1). The graph contains an edge (d_j, C_k) if block C_k can be used to reconstruct document d_j. Note that even if the graph contains (d_j, C_k), it may still be possible to reconstruct d_j from other blocks excluding C_k. Document nodes in Dagster's entanglement graph have an out-degree $c + 1$, and those in Tangler's have out-degree four. Entangled documents share one or more server blocks. In Figure 4.1, documents d_1 and d_n are entangled because they share server block C_1; meanwhile, documents d_1 and d_2 are not entangled.

This shared-block notion of entanglement has several drawbacks. Even if document d_j is entangled with a specific document, it may still be possible to delete d_j

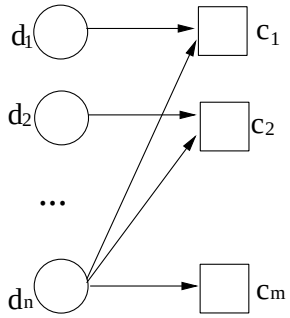

Figure 4.1: An entanglement graph is a bipartite graph from the set of documents to the set of server blocks. An edge (d_j, C_k) is in the graph if server block C_k can be used to reconstruct document d_j.

from the server without affecting that particular document. For example, knowing that d_n is entangled with d_1 (as in Figure 4.1), and that d_1 is owned by some Very Important Person, may give solace to the owner of d_n, who might assume that no adversary would dare incur the wrath of the VIP merely to destroy d_n. But in the situation depicted in the figure, the adversary can still delete server blocks C_2 and C_m and corrupt d_n but not d_1.

Moreover, the user does not get to choose the documents to be entangled with his document; these documents are chosen randomly. While destroying a user's document is likely to destroy some others, there are no specific other documents that will necessarily be destroyed. If few other documents get destroyed on average, the small risk of accidentally corrupting an important document will be unlikely to deter the adversary from tampering with data.

We now derive an upper bound on how many documents get destroyed if we delete a random document from a Dagster or Tangler server. We consider a restricted adversary, who randomly chooses several blocks of an arbitrary document (one block in Dagster; two in Tangler) and overwrites them with zeroes. Intuitively, one might expect that the earlier the document was uploaded onto the server, the more documents it will be entangled with and the more other documents will be

destroyed.

This turns out to be the case, as we prove for any specific target document in Lemmas 8 and 9. In these two lemmas, we bound the expected number of documents lost when deleting the j-th of n documents. The effects of deleting a document chosen uniformly at random, stated in Claims 10 and 11, are computed by averaging these bounds over all documents.

Without loss of generality, we assume that documents are numbered in the order in which they were uploaded; namely, for all $1 \leq j < n$, document d_j was uploaded to the server before d_{j+1}.

Lemma 8 *In a Dagster server with $n_0 = O(1)$ initial blocks and n documents, where each document is linked with c pre-existing blocks, deleting a random block of document d_j $(1 \leq j \leq n)$ destroys on average*

$$O\left(c \log\left(\frac{n}{j}\right)\right)$$

other documents.

Proof: Altogether, there are $n_0 + n$ blocks stored on the server: n_0 initial blocks and n data blocks. We label the data blocks $C_1, \ldots, C_n$. The initial blocks exist on the server before any data blocks have been added. We label them $C_{-n_0+1}, \ldots, C_0$.

Every document d_j consists of c pre-existing or "old" blocks[2] and a "new" data block C_j that is computed during the entanglement stage. Consider an adversary who destroys a random block C_i of d_j. This will destroy d_j, but it will also destroy any documents with outgoing edges to C_i in the entanglement graph. We would like to compute the number of such documents, N_i.

[2]These may be either initial blocks or data blocks of documents added earlier than d_j (*i.e.*, d_k with $k < j$).

If C_i is a data block (*i.e.*, $i \geq 1$), then

$$\begin{aligned} E[N_i] &= \sum_{k=i}^{n} \Pr[d_k \text{ has an edge to } C_i] \\ &= 1 + \sum_{k=i+1}^{n} \left(1 - \binom{k-2+n_0}{c} / \binom{k-1+n_0}{c}\right) \\ &< 1 + c \sum_{j=i+n_0}^{n+n_0-1} \frac{1}{j} \\ &= 1 + c(h_{n+n_0-1} - h_{i+n_0-1}) \\ &= O\left(c \log\left(\frac{n}{i}\right)\right) \text{ under the assumption that } n_0 \text{ is a constant.} \end{aligned}$$

Meanwhile, if C_i is an initial block (*i.e.*, $i < 1$), it can be linked by any of the documents:

$$\begin{aligned} E[N_i] &= \sum_{k=1}^{n} \Pr[d_k \text{ has an edge to } C_i] \\ &= O(c \log n). \end{aligned}$$

The number of documents deleted on average when the adversary destroys a *random* block of d_j is

$$\begin{aligned} N_{avg} &= \frac{1}{j+n_0} \left(\sum_{i=-n_0}^{j} E[N_i] \right) \\ &< \frac{1}{j} \left(O(c \log n) + \sum_{i=1}^{j} O\left(c \log\left(\frac{n}{i}\right)\right) \right) \end{aligned} \tag{4.1}$$

We can use Stirling's formula to bound the leading term in (4.1):

$$\begin{aligned} \sum_{i=1}^{j} O\left(c \log\left(\frac{n}{i}\right)\right) &= O\left(c \log\left(\prod_{i=1}^{j} \frac{n}{i}\right)\right) \\ &= O\left(cj \log\left(\frac{n}{j/e}\right)\right) \end{aligned}$$

$$= O\left(cj \log\left(\frac{n}{j}\right)\right).$$

The lemma follows.

Lemma 9 *In a Tangler server with $n_0 = O(1)$ initial blocks and n documents, deleting two random blocks of document d_j $(1 \leq j \leq n)$ destroys on average*

$$O\left(\frac{1}{j}\right)$$

other documents.

Proof: The server contains $n_0 + 2n$ blocks, n_0 of which are initial and $2n$ are data blocks. We label the blocks as in the proof of Claim 10. The initial blocks are $C_{-n_0+1}, \ldots, C_0$ and the data blocks are $C_1, C_2, \ldots, C_{2n-1}, C_{2n}$.

In Tangler, every document d_j consists of two *old* blocks of pre-existing documents and two *new* blocks C_{2j-1} and C_{2j}, computed during the entanglement stage. Suppose an adversary deletes any two out of four blocks comprising d_j; call these blocks C_i, C_t. Then any document d_k $(k \neq j)$ that contains both C_i and C_t (*i.e.*, has edges outgoing to C_i and C_t in the entanglement graph), will also get destroyed. We would like to compute the number of such documents, N_{avg}.

In our analysis, we consider whether deleted blocks C_i, C_t are new or old to d_j and d_k. We distinguish between five cases:

Case 1: C_i, C_t are old to both d_j and d_k. Then the number of deleted documents is

$$\sum_{k=1}^{j-1} \frac{1}{\binom{2j-2+n_0}{2}} + \sum_{k=j+1}^{n} \frac{1}{\binom{2k-2+n_0}{2}}.$$

Case 2: C_i, C_t are old to d_j. However, only one of them is old to d_k, while the other is new to d_k. Note that, in this case, we must have $k < j$.

$$\sum_{k=1}^{j-1} \frac{4}{\binom{2j-2+n_0}{2}}.$$

Case 3: C_i, C_t are old to d_j, but new to d_k. Note that, in this case, we must also have $k < j$.

$$\sum_{k=1}^{j-1} \frac{1}{\binom{2j-2+n_0}{2}}.$$

Case 4: One of C_i, C_t is old to d_j, the other is new to d_j. Note that both C_i and C_t must be old to d_k (because otherwise we would have $k < j$, which implies that the block in C_i, C_t that is new to d_j is not linked to d_k, which further implies that d_k will not get deleted).

$$\sum_{k=j+1}^{n} \frac{4}{\binom{2k-2+n_0}{2}}.$$

Case 5: C_i, C_t are new to d_j. In this case, both C_i and C_t must be old to d_k (for the same reason as in Case 4).

$$\sum_{k=j+1}^{n} \frac{1}{\binom{2k-2+n_0}{2}}.$$

Summing up the five cases gives us the total number of documents destroyed:

$$N_{avg} = \frac{6(j-1)}{\binom{2j-2+n_0}{2}} + \sum_{k=j+1}^{n} \frac{6}{\binom{2k-2+n_0}{2}}$$

$$
\begin{aligned}
&< \frac{6}{2j-3} + \sum_{k=j+1}^{n} \frac{3}{(k-\frac{3}{2})^2} \\
&< \frac{6}{2j-3} + \int_{j}^{n} \frac{3}{(x-\frac{3}{2})^2}\, dx \\
&= \frac{12}{2j-3} - \frac{6}{2n-3} \\
&= O\left(\frac{1}{j}\right) \text{ for large } n.
\end{aligned}
$$

Using Lemma 8, we can show that destroying a typical document in Dagster affects few other documents on average:

Claim 10 *In a Dagster server with n documents, where each document is linked with c pre-existing blocks, deleting a block of a random document destroys on average $O(c)$ other documents.*

Proof: Suppose the server contains n documents, each document linked with c server blocks. Then, Lemma 8 tells us that deleting document d_j $(1 \leq j \leq n)$ affects $O\left(c \log\left(n/j\right)\right)$ other documents. Therefore, deleting a typical d_j will affect

$$
\begin{aligned}
\frac{1}{n}\sum_{j=1}^{n} O\left(c \log\left(\frac{n}{j}\right)\right) &= O\left(\frac{c}{n} \log\left(\prod_{j=1}^{n} \frac{n}{j}\right)\right) \\
&= O\left(\frac{c}{n} \log\left(\frac{n^n}{(n/e)^n}\right)\right) \\
&= O(c)
\end{aligned}
$$

documents, as claimed.

We have a similar result for destroying a typical document in Tangler:

Claim 11 *In a Tangler server with n documents, deleting two blocks of a random document destroys on average $O\left(\frac{\log n}{n}\right)$ other documents.*

The proof of Claim 11 is immediate from Lemma 8, so we do not give it here.

Even a small chance of destroying an important document will deter tampering to some extent, but some tamperers might be willing to run that risk. Still more troubling is the possibility that the tamperer might first flood the system with junk documents, so that almost all real documents were entangled only with junk. Since our bounds show that destruction of a typical document will on average affect only a handful of others in Dagster and almost none in Tangler, we will need stronger entanglement mechanisms if entanglement is to deter tampering by itself.

4.3 Our Model

In Subsection 4.3.1, we start by giving a basic framework for modeling systems such as Dagster and Tangler that entangle data. Specializing the general framework gives specific system models, differentiated by the choice of recovery algorithms and restrictions placed on the adversary. We discuss them in Section 4.3.2.

As we mentioned, Dagster and Tangler use many worthwhile techniques in conjunction with entanglement to provide censorship-resistance. Our model abstracts away many such details of storage and recovery processes. We concentrate on a single entanglement operation performed by these systems, which takes documents of a finite group of users and intertwines these documents to form a common store. In practice, the server contents would be computed as an aggregation of common stores from multiple entanglement operations.

4.3.1 Basic Framework

Our model consists of an *initialization phase*, in which keys are generated and distributed to the various participants in the system; an *entanglement phase*, in which

the individual users' data are combined into a common store; a *tampering phase*, in which the adversary corrupts the store; and a *recovery phase*, in which the users attempt to retrieve their data from the corrupted store using one or more recovery algorithms. For simplicity of notation, we number the users $\{1, \ldots, n\}$, where every user i possesses a document d_i that he wants to publish.

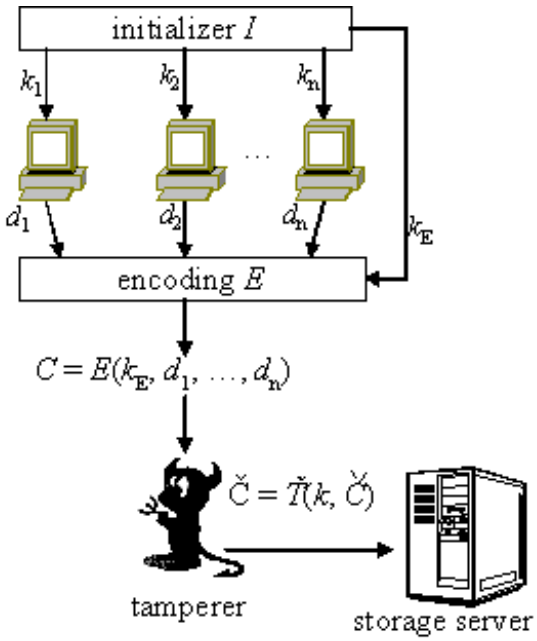

Figure 4.2: Initialization, entanglement, and tampering stages.

An encoding scheme consists of three probabilistic Turing machines, (I, E, R), that run in time polynomial in the size of their inputs and a security parameter s. The first of these, the *initialization algorithm* I, hands out the keys used in the encoding and recovery phases. The second, the *encoding algorithm* E, combines the users' data into a common store using the encoding key. The third, the *recovery algorithm* R, attempts to recover each user's data using the appropriate recovery key.

Acting against the encoding scheme is an adversary $(\check{I}, \check{T}, \check{R})$, which also consists of three probabilistic polynomial-time Turing machines . The first is an *adversary-*

initialization algorithm $\check{I}$; like the good initializer I, the evil $\check{I}$ is responsible for generating keys used by other parts of the adversary during the protocol. The second is a *tampering algorithm* $\check{T}$, which modifies the common store. The third is a *non-standard recovery algorithm* $\check{R}$, which may be used by some or all of the users to recover their data from the modified store.

We assume that $\check{I}$, $\check{T}$ and $\check{R}$ are chosen after I, E, and R are known but that a single fixed $\check{I}$, $\check{T}$, and $\check{R}$ are used for arbitrarily large values of s and n.

Given an encoding scheme (I, E, R) and an adversary $(\check{I}, \check{T}, \check{R})$, the *storage protocol* proceeds as follows (see also Figure 4.2):

1. *Initialization.* The initializer I generates a combining key k_E used by the encoding algorithm and recovery keys $k_1, k_2, \ldots k_n$, where each key k_i is used by the recovery algorithm to recover the data for user i. At the same time, the adversary initializer $\check{I}$ generates the shared key $\check{k}$ for $\check{T}$ and $\check{R}$.

$$k_E, k_1, k_2, \ldots k_n \leftarrow I(1^s, n),$$

$$\check{k} \leftarrow \check{I}(1^s, n).$$

2. *Entanglement.* The encoding algorithm E computes the combined store C from the combining key k_E and the data d_i:

$$C \leftarrow E(k_E, d_1, d_2, \ldots d_n).$$

3. *Tampering.* The tamperer $\check{T}$ alters the combined store C into $\check{C}$:

$$\check{C} \leftarrow \check{T}(\check{k}, C).$$

4. *Recovery.* The users attempt to recover their data. User i applies his recovery algorithm R_i to k_i and the changed store $\check{C}$. Each R_i could be either the standard recovery algorithm R, supplied with the encoding scheme, or the non-standard algorithm $\check{R}$, supplied by the adversary, depending on the choice of the model.

$$d'_i \leftarrow R_i(k_i, \check{C})$$

We say that user i *recovers* his data if the output of R_i equals d_i.

4.3.2 Adversary Classes

We divide our model on two axes: one bounding the users' choices of reconstruction algorithms and the other bounding the adversary's power to modify the data store. With respect to recovery algorithms, we consider three variants on the basic framework (listed in order of increasing power given to the adversary):

- In the *standard-recovery-algorithm model*, the users are restricted to a single standard recovery algorithm R, supplied by the system designer. Formally, this means $R_i = R$ for all users i; The adversary's recovery algorithm $\check{R}$ is not used. This is the model used to analyze Dagster and Tangler.

- In the *public-recovery-algorithm model*, the adversary not only modifies the combined store, but also supplies a single non-standard recovery algorithm $\check{R}$ to all of the users. Formally, we have $R_i = \check{R}$ for each i. The original recovery algorithm R is not used.[3] We call this an *upgrade attack* by analogy to the real life situation of a company changing the data format of documents processed

[3]Though it may seem unreasonable to prevent users from choosing the original recovery algorithm R, any R can be rendered useless in practice by superencrypting the data store and distributing the decryption key only with the adversary's $\check{R}$. We discuss this issue further in Subsection 4.5.2.

by its software and distributing a new version of the software to read them. We believe such an attack is a realistic possibility, because most self-interested users will be happy to adopt a new recovery algorithm if it offers new features or performance, or if the alternative is losing their data.

- In the *private-recovery-algorithm model*, the adversary may choose to supply the non-standard recovery algorithm $\check{R}$ to only a subset of the users. The rest continue to use the standard algorithm R. Formally, this model is a mix of the previous two models: $R_i = R$ for some i and $R_i = \check{R}$ for others.

We also differentiate between two types of tamperers:

- An *arbitrary tamperer* can freely corrupt the data store and is not restricted in any way. Most real-life systems fit into this category as they place no restrictions on the tamperer.

- A *destructive tamperer* can only apply a transformation to the store whose range of possible outputs is substantially smaller than the set of inputs. The destructive tamperer can superimpose its own encryption on the common store, transform the store in arbitrary ways, and even add additional data, provided that the cumulative effect of all these operations is to decrease the entropy of the data store. Though a destructive tampering assumption may look like an artificial restriction, it subsumes natural models of block deletion or corruption, and either it or some similar assumption is needed to achieve all-or-nothing integrity in the private-recovery-algorithm model.

An *adversary class* specifies what kind of tamperer $\check{T}$ is and which users, if any, receive $\check{R}$ as their recovery algorithm. Altogether, we consider 6 ($= 3 \times 2$) adversary classes, each corresponding to a combination of constraints on the tamperer and the recovery algorithms.

4.4 Dependency and All-or-Nothing Integrity

We now give our definition of document dependency for a particular encoding scheme and adversary class. We first discuss some basic definitions and assumptions in Subsection 4.4.1. Our strong notions of entanglement, called *dependency* and *all-or-nothing integrity*, are defined formally in Subsection 4.4.2.

4.4.1 Preliminaries

Fix an encoding (I, E, R), an adversary $A = (\check{I}, \check{T}, \check{R})$, and the recovery algorithm R_i for each user i. An *execution* of the resulting system specifies the inputs k_i and d_i to E, the output of E, the tamperer's input $\check{k}$ and output $\check{C}$, and the output of the recovery algorithm R_i ($R(k_i, \check{C})$ or $\check{R}(\check{k}, k_i, \check{C})$ as appropriate) for each user. The set of possible executions of the storage system is assigned probabilities in the obvious way: the probability of an execution is taken over the inputs to the storage system and the coin tosses of the encoding scheme and the adversary. It will be convenient to consider multiple adversaries with a fixed encoding scheme. In this case, we use $\Pr_A(Q)$ to denote the probability that an event Q occurs when A is the adversary.

During an execution of the storage system, the tamperer alters the combined store from C into $\check{C}$. As a result, some users end up recovering their documents while others do not. The *recovery vector* of an execution specifies which documents were successfully recovered in that execution. Formally:

Definition 12 *The* recovery vector *of execution α, denoted $\vec{\rho}(\alpha)$, is a bit vector*

$$\vec{\rho}(\alpha) = (\rho_1, \rho_2, \ldots, \rho_n),$$

where

$$\rho_i = \begin{cases} 1 & \text{if } R_i(k_i, \check{C}) = d_i, \\ 0 & \text{otherwise} \end{cases}$$

To illustrate, suppose that the server contains three documents: d_1, d_2, and d_3. If in execution α we recover only documents d_1 and d_2, we then have: $\vec{\rho}(\alpha) = 110$.

When we think of α as a random variable (having fixed a particular encoding algorithm and adversary), we will use $\vec{r}$ as a shorthand for the random variable $\vec{\rho}(\alpha)$.

4.4.2 Our Notions of Entanglement

In Section 4.2, we observed that the block-sharing notion of entanglement provided by Dagster and Tangler does not by itself provide strong security guarantees. Two documents may be entangled in this sense even though it is still possible to delete one of them without affecting the other. This motivates us to propose the notion of *document dependency*. Document dependency formalizes the idea that "if my data depends on yours, I can't get my data back if you can't." In this way, the fates of specific documents become linked together: specifically, if document d_i depends on document d_j, then whenever d_j cannot be recovered neither can d_i. Formally:

Definition 13 *A document* d_i depends on *a document* d_j *with respect to a class of adversaries* $\mathcal{A}$*, denoted* $d_i \overset{\mathcal{A}}{\hookrightarrow} d_j$*, if, for all adversaries* $A \in \mathcal{A}$*,*

$$\Pr_A[r_i = 0 \vee r_j = 1] \geq 1 - \epsilon,$$

where ϵ *is negligible in the security parameter* s*.*

Remark: Hereafter, ϵ refers to a negligible function of the security parameter s.

The ultimate form of dependency is *all-or-nothing integrity*. Intuitively, a storage

system is all-or-nothing if either every user i recovers his data or no user does:

Definition 14 *A storage system is* all-or-nothing *with respect to a class of adversaries $\mathcal{A}$ if, for all $A \in \mathcal{A}$,*

$$\Pr_A[\vec{r} = 0^n \vee \vec{r} = 1^n] \geq 1 - \epsilon.$$

It is easy to show that

Lemma 15 *A storage system is all-or-nothing with respect to a class of adversaries $\mathcal{A}$ if and only if, for all users i, j, $d_i \overset{\mathcal{A}}{\hookrightarrow} d_j$.*

Proof: Fix an adversary in $\mathcal{A}$. Let E be the event that an execution of the storage system is not all-or-nothing, and F_{ij} the event that document d_i was recovered in an execution and d_j was not. Then $E = \{\vec{r} \neq 0^n \wedge \vec{r} \neq 1^n\}$ and $F_{ij} = \{r_i = 1 \wedge r_j = 0\}$.

($\Rightarrow$) : If the system is all-or-nothing, then $\Pr[E] < \epsilon$. Clearly, for all i, j, we have $F_{ij} \subseteq E$, which means $\Pr[F_{ij}] \leq \Pr[E] < \epsilon$. This in turn implies $d_i \overset{\mathcal{A}}{\hookrightarrow} d_j$.

($\Leftarrow$) : If for all i, j, $d_i \overset{\mathcal{A}}{\hookrightarrow} d_j$, then $\Pr[F_{ij}] < \epsilon$. We can choose $\epsilon < \epsilon'/n^2$ for a negligible ϵ'.

Notice that $E \subseteq \bigcup_{i,j} F_{ij}$. Therefore, $\Pr[E] \leq \sum_{i,j} \Pr[F_{ij}] < n^2\epsilon < \epsilon'$. Hence, $\Pr[E^c] \geq 1 - \epsilon'$ and so the storage system is all-or-nothing.

All-or-nothing integrity is a very strong property. In some models, we may not be able to achieve it, and we will accept a weaker property called *symmetric recovery*. Symmetric recovery requires that all users recover their documents with equal probability:

Definition 16 *A storage system has* symmetric recovery *with respect to a class of adversaries $\mathcal{A}$ if, for all $A \in \mathcal{A}$ and all users i and j,*

$$\Pr_A[r_i = 1] = \Pr_A[r_j = 1].$$

Symmetric recovery says nothing about what happens in particular executions. For example, it is consistent with the definition for exactly one of the data items to be recovered in every execution, as long as the adversary cannot affect which data item is recovered. This is not as strong a property as all-or-nothing integrity, but it is the best that can be done in some cases.

4.5 Possibility and Impossibility Results

The possibility of achieving *all-or-nothing integrity* (abbreviated AONI) depends on the class of adversaries we consider. In Subsections 4.5.1 through 4.5.3, we consider adversaries with an *arbitrary tamperer*. In the *standard-recovery-algorithm model*, a simple application of Message Authentication Codes (MACs) achieves all-or-nothing integrity. In the *public-recovery-algorithm model*, all-or-nothing integrity is no longer possible. The best that can be done is to prevent the adversary from targeting specific users by hiding the location of each user's data within the store. In the *private-recovery-algorithm model*, even the weak guarantees of the public-recovery-algorithm model are no longer possible, because the adversary can *superencrypt* the data store and refuse to distribute the decryption key to users he doesn't like.

In Subsection 4.5.4, we look at adversaries with a *destructive tamperer*. We give a simple interpolation scheme that achieves all-or-nothing integrity for a destructive tamperer in all three recovery models.

4.5.1 Possibility of AONI in the Standard-Recovery-Algorithm Model

In the standard-recovery-algorithm model, all users use the standard recovery algorithm R; that is $R_i = R$ for all users i. Both Dagster and Tangler assume this model.

This model allows a very simple mechanism for all-or-nothing integrity based on Message Authentication Codes (MACs).[4] The intuition behind this mechanism is that the encoding algorithm E simply tags the data store with a MAC using a key known to all the users, and the recovery algorithm R returns an individual user's data only if the MAC on the entire database is valid.

We now give an encoding scheme (I, E, R) based on a MAC scheme (GEN, TAG, VER):

Initialization The initialization algorithm I computes $k_{MAC} = GEN(1^s)$. It then returns an encoding key $k_E = k_{MAC}$ and recovery keys $k_i = (i, k_{MAC})$.

Entanglement The encoding algorithm E generates an n-tuple $m = (d_1, d_2, \ldots, d_n)$ and returns $C = (m, \sigma)$ where $\sigma = TAG(k_{MAC}, m)$.

Recovery The standard recovery algorithm R takes as input a key $k_i = (i, k_{MAC})$ and the (possibly modified) store $\check{C} = (\check{m}, \check{\sigma})$. It returns $\check{m}_i$ if $VER(k_{MAC}, \check{m}, \check{\sigma}) =$ accept and returns a default value $\perp$ otherwise.

The following theorem states that this encoding scheme achieves all-or-nothing

[4] Recall that a MAC consists of three algorithms (GEN, TAG, VER) (see [64]). A key generator GEN on input 1^s outputs an s-bit key k_{MAC}. A tagging algorithm TAG on input k_{MAC} and message m ($|m| \leq s^c$) computes a signature σ. A verification algorithm VER can be used to check if σ is a valid signature on m. It has the property that $VER(k_{MAC}, m, TAG(k_{MAC}, m)) = accept$ for all m.

We require a MAC to be *existentially unforgeable* under *chosen message attacks*. This means there is no polynomial time forger F that generates a new message-signature pair (m', σ') that is accepted by VER with probability exceeding $O(s^{-c})$ for any $c > 0$, even if F is given a sample of valid message-signature pairs (m_i, σ_i), where m_i is chosen by the adversary.

integrity with standard recovery algorithms:

Theorem 17 *Let (GEN, TAG, VER) be a MAC scheme that is existentially unforgeable against chosen message attacks, and let (I, E, R) be an encoding scheme based on this MAC scheme as above. Let $\mathcal{A}$ be the class of adversaries that does not provide non-standard recovery algorithms $\check{R}$. Then there exists some minimum s_0 such that for any security parameter $s \geq s_0$ and any inputs $d_1, \ldots, d_n$ with $\sum |d_i| \leq s$, (I, E, R) is all-or-nothing with respect to $\mathcal{A}$.*

Proof: Fix some $c > 0$. Recall that the adversary changes the combined store from $C = (m, \sigma)$ to $\check{C} = (\check{m}, \check{\sigma})$. We consider two cases, depending on whether or not $\check{m} = m$.

In the first case, $\check{m} = m$. Suppose $R(k_i, \check{C}) = d_i$ but $R(k_j, \check{C}) \neq d_j$. Then $R(k_j, \check{C}) = \perp$, which implies that $V(k_{MAC}, m, \check{\sigma}) \neq$ accept when computed by $R(k_j, \check{C})$ and thus that $\check{\sigma} \neq \sigma$. But $R(k_i, \check{C}) = d_i$ only if $V(k_{MAC}, m, \check{\sigma}) =$ accept when computed by $R(k_i, \check{C})$. It follows that $(m, \check{\sigma})$ is a message-MAC pair not equal to (m, σ) that V accepts in the execution of $R(k_i, \check{C})$; by the security assumption this occurs for a particular execution of V only with probability $O(s^{-c'})$ for any fixed c'. If we choose c' and s_0 so that the $O(s^{-c'})$ term is smaller than $\frac{1}{2n} s^{-c}$ for $s \geq s_0$, then the probability that *any* of the n executions of V in the recovery stage accepts $(m, \check{\sigma})$ in some case where $m = \check{m}$, is bounded by $\frac{1}{2} s^{-c}$.

In the second case, $m \neq \check{m}$. Now $(\check{m}, \check{\sigma})$ is a message-MAC pair not equal to (m, σ). If every execution of V rejects $(\check{m}, \check{\sigma})$, then all $R(d_i, \check{C})$ return $\perp$ and the execution has a recovery vector 0^n. The only bad case is when at least one execution of V erroneously accepts $(\check{m}, \check{\sigma})$. But using the security assumption and choosing c', s_0 as in the previous case, we again have that the probability that V accepts $(\check{m}, \check{\sigma})$ in any of the n executions of R is at most $\frac{1}{2} s^{-c}$.

Summing the probabilities of the two bad cases gives us the desired bound: $\Pr_A[\vec{r} = 0^n \vee \vec{r} = 1^n] > 1 - s^{-c}$.

4.5.2 Impossibility of AONI in the Public and Private-Recovery-Algorithm Models

In both these models, the adversary modifies the common store and distributes a non-standard recovery algorithm $\check{R}$ to the users (either to all users or only to a few select accomplices). Let us begin by showing that all-or-nothing integrity cannot be achieved consistently in either case:

Theorem 18 *For any encoding scheme (I, E, R), if $\mathcal{A}$ is the class of adversaries providing non-standard recovery algorithms $\check{R}$, then (I, E, R) is not all-or-nothing with respect to $\mathcal{A}$.*

Proof: Let the adversary initializer $\check{I}$ be a no-op and let the tamperer $\check{T}$ be the identity transformation. We will rely entirely on the non-standard recovery algorithm to destroy all-or-nothing integrity.

Let $\check{R}$ flip a biased coin that comes up tails with probability $1/n$, and return the result of running R on its input if the coin comes up heads and $\perp$ if the coin comes up tails. Then exactly one document is not returned with probability $n \cdot (1/n) \cdot (1 - 1/n)^{n-1}$, which converges to $1/e$ in the limit. Because this document is equally likely to be any of the n documents by symmetry, we get each of the recovery vectors described in the theorem with a non-negligible probability that converges to $1/en$.

The outcome is all-or-nothing only if all instances of $\check{R}$ flip the same way, which occurs with probability $\Pr_A[\vec{r} = 0^n \vee \vec{r} = 1^n] < 1 - 1/en$.

The proof of Theorem 18 is rather trivial, which suggests that letting the adversary substitute an error-prone recovery algorithm in place of the standard one gives

the adversary far too much power. But it is not at all clear how to restrict the model to allow the adversary to provide an improved recovery algorithm without allowing for this particular attack.

One possibility would be to allow users to choose between applying the original recovery algorithm and the adversary's new and improved version; but in practice this approach is easily defeated by a tamperer $\check{T}$ that encrypts C (which renders $\check{C}$ unusable as input to R) coupled with an error-prone $\check{R}$ that reverses the encryption (when its coin comes up heads) before applying R.

A more sophisticated approach would be to allow R to analyze $\check{R}$ to attempt to undo whatever superencryption may have been performed and extract a recovery algorithm that works all the time. Unfortunately, this approach depends on being able to extract useful information about the workings of an arbitrary Turing machine. While it has been shown that program obfuscation is impossible in general [11], even in a specialized form this operation is likely to be very difficult, especially if the random choice to decrypt incorrectly is not a single if-then test but is the result of accumulating error distributed throughout the computation of $\check{R}$.

On the other hand, we do not know of any general mechanism to ensure that no useful information can be gleaned from $\check{R}$, and it is not out of the question that there is an encoding so transparent that no superencryption can disguise it for sufficiently large inputs, given that both $\check{R}$ and the adversary's key $\check{k}$ are public.

4.5.3 Possibility of Symmetric Recovery in the Public-Recovery-Algorithm Model

As we have seen, if we place no restrictions on the tamperer, it becomes impossible to achieve all-or-nothing integrity in the public-recovery-algorithm model. We now

show that we can still achieve symmetric recovery.

Because we cannot prevent mass destruction of data, we will settle for preventing targeted destruction. The basic intuition is that if the encoding process is symmetric with respect to permutations of the data, then neither the tampering algorithm nor its partner, the non-standard recovery algorithm, can distinguish between different inputs. Symmetry in the encoding algorithm is not difficult to achieve and basically requires not including any positional information in the keys or the representation of data in the common store. One example of a symmetric encoding is a trivial mechanism that tags each input d_i with a random k_i and then stores a sequence of (d_i, k_i) pairs in random order.

Symmetry in the data is a stronger requirement. Here, we assume that users' documents d_i are independent and identically distributed (i.i.d.) random variables. If documents are not i.i.d (in particular, if they are fixed), we can use a simple trick to make them appear i.i.d.: Each user i picks a small number r_i independently and uniformly at random, remembers the number, and computes $d'_i = d_i \oplus G(r_i)$, where G is a pseudorandom generator. The users can then store documents d'_i $(1 \leq i \leq n)$ instead of the original documents d_i. To recover d_i, user i would retrieve d'_i from the server and compute $d_i = d'_i \oplus G(r_i)$.

We shall need a formal definition of symmetric encodings:

Definition 19 *An encoding scheme* (I, E, R) *is* symmetric *if, for any* s *and* n*, any inputs* $d_1, d_2, \ldots d_n$*, and any permutation* π *of the indices* 1 *through* n*, if the joint distribution of* $k_1, k_2, \ldots, k_n$ *and* C *in executions with user inputs* $d_1, d_2, \ldots d_n$ *is equal to the joint distribution of* $k_{\pi_1}, k_{\pi_2}, \ldots, k_{\pi_n}$ *and* C *in executions with user inputs* $d_{\pi_1}, d_{\pi_2}, \ldots d_{\pi_n}$*.*

Using this definition, it is easy to show that any symmetric encoding gives sym-

metric recovery:

Theorem 20 *Let (I, E, R) be a symmetric encoding scheme. Let $\mathcal{A}$ be a class of adversaries as in Theorem 18. Fix s and n, and let $d_1, \ldots, d_n$ be random variables that are independent and identically distributed. Then (I, E, R) has symmetric recovery with respect to $\mathcal{A}$.*

Proof: Fix i and j. From Definition 19 we have that the joint distribution of the k_i and C is symmetric with respect to permutation of the user indices; in particular, for any fixed d, S and x,

$$\Pr[C = S, k_i = x \,|\, d_i = d] = \Pr[C = S, k_j = x \,|\, d_j = d]. \tag{4.2}$$

We also have, from the assumption that the d_i are i.i.d.,

$$\Pr[d_i = d] = \Pr[d_j = d]. \tag{4.3}$$

Using (4.2) and (4.3), we get

$$\begin{aligned}
&\Pr[\check{R}(\check{k}, k_i, \check{T}(C)) = d_i] \\
&\quad = \sum_{x,S,d} \Pr[\check{R}(\check{k}, x, \check{T}(S)) = d] \Pr[C = S, k_i = x, d_i = d] \\
&\quad = \sum_{x,S,d} \Pr[\check{R}(\check{k}, x, \check{T}(S)) = d] \Pr[C = S, k_i = x \,|\, d_i = d] \Pr[d_i = d] \\
&\quad = \sum_{x,S,d} \Pr[\check{R}(\check{k}, x, \check{T}(S)) = d] \Pr[C = S, k_j = x \,|\, d_j = d] \Pr[d_j = d] \\
&\quad = \Pr[\check{R}(\check{k}, k_j, \check{C}) = d_j].
\end{aligned}$$

This is simply another way of writing $\Pr_A[r_i = 1] = \Pr_A[r_j = 1]$.

4.5.4 Possibility of AONI for Destructive Adversaries

Unfortunately, neither all-or-nothing integrity nor symmetric recovery can be achieved in the private-recovery-algorithm model for an arbitrary tamperer. The adversary can always superencrypt the data store and distribute a useless recovery algorithm to some users that refuses to return the data. We need to place some additional restrictions on the adversary.

A tampering algorithm $\check{T}$ is *destructive* if the range of $\check{T}$ when applied to an input domain of m distinct possible data stores has size less than m. The amount of destructiveness is measured in bits: if the range of $\check{T}$ when applied to a domain of size m has size r, then $\check{T}$ destroys $\lg m - \lg r$ bits of entropy. Note that it is not necessarily the case that the outputs of $\check{T}$ are smaller than its inputs; it is enough that there be fewer of them.

Below, we describe a particular encoding, based on polynomial interpolation, with the property that after a sufficiently destructive tampering, the probability that *any* recovery algorithm can reconstruct a particular d_i is small. While this is trivially true for an unrestrained tamperer that destroys all $\lg m$ bits of the common store, our scheme requires only that with n documents the tamperer destroy slightly more than $n \lg(n/\epsilon)$ bits before the probability that any of the data can be recovered drops below ϵ (a formal statement of this result is found in Corollary 22). Because n counts only the number of users and not the size of the data, for a fixed population of users the number of bits that can be destroyed before all users lose their data is effectively a constant independent of the size of the store being tampered with.

The encoding scheme is as follows. It assumes that each data item can be encoded as an element of Z_p, where p is a prime of roughly s bits.

Initialization The initialization algorithm I chooses $k_1, k_2, \ldots k_n$ independently and

uniformly at random *without replacement* from Z_p. It sets $k_E = (k_1, k_2, \ldots, k_n)$ and then returns $k_E, k_1, \ldots k_n$.

Entanglement The encoding algorithm E computes, using Lagrange interpolation, the coefficients $c_{n-1}, c_{n-2}, \ldots c_0$ of the unique degree $(n-1)$ polynomial f over Z_p with the property that $f(k_i) = d_i$ for each i. It returns $C = (c_{n-1}, c_{n-2}, \ldots c_0)$.

Recovery The standard recovery algorithm R returns $f(k_i)$, where f is the polynomial whose coefficients are given by C.

Intuitively, the reason the tamperer cannot remove too much entropy without destroying all data is that it cannot identify which points $d = f(k)$ correspond to actual user keys. When it maps two polynomials f_1 and f_2 to the same corrupted store $\check{C}$, the best that the non-standard recovery algorithm can do is return one of $f_1(k_i)$ or $f_2(k_i)$ given a particular key k_i. But if too many polynomials are mapped to the same $\check{C}$, the odds that $\check{R}$ returns the value of the correct polynomial will be small.

A complication is that a particularly clever adversary could look for polynomials whose values overlap; if $f_1(k) = f_2(k)$, it doesn't matter which f the recovery algorithm picks. But here we can use that fact that two degree $(n-1)$ polynomials cannot overlap in more than $(n-1)$ places without being equal. This limits how much packing the adversary can do.

As in Theorem 20, we assume that the user inputs $d_1, \ldots, d_n$ are chosen independently and have identical distributions. We make a further assumption that each d_i is chosen uniformly from Z_p. This is necessary to ensure that the resulting polynomials span the full p^n possibilities.[5]

[5]The assumption that the documents are i.i.d. does not constraint the applicability of our results much, because the technique to get rid of it described in Section 4.5.2 can also be used here.

Under these conditions, sufficiently destructive tampering prevents recovery of any information with high probability. We will show an accurate but inconvenient bound on this probability in Claim 21 and give a cruder but more useful statement of the bound in Corollary 22.

Claim 21 *Let (I, E, R) be defined as above. Let $A = (\check{I}, \check{T}, \check{R})$ be an adversary where $\check{T}$ is destructive: for a fixed input size and security parameter, there is a constant M such that for each key $\check{k}$,*

$$|\{\check{T}(\check{k}, f)\}| \leq M,$$

where f ranges over the possible store values, i.e. over all degree-$(n-1)$ polynomials over Z_p. If the d_i are drawn independently and uniformly from Z_p, then the probability that at least one user i recovers d_i using $\check{R}$ is

$$\Pr_A[\vec{r} \neq 0^n] < \frac{2n^2 + nM^{1/n}}{p}, \tag{4.4}$$

even if all users use $\check{R}$ as their recovery algorithm.

Proof: Condition on $\check{k}$ and the outcome of all coin-flips used by $\check{T}$ and $\check{R}$. Then, there are exactly $p^n\binom{p}{n}$ possible executions, each of equal probability, determined by the p^n choices for the d_i and the $\binom{p}{n}$ choices for the k_i. For each i, we will show that the number of these executions in which $\check{R}(\check{k}, k_i, \check{C}) = d_i$ is small.

For each degree-$(n-1)$ polynomial f, define f^* to be the function mapping each k in Z_p to $\check{R}(\check{k}, k, \check{T}(\check{k}, f))$. Note that f^* is deterministic given that we are conditioning on $\check{k}$ and all coin-flips in $\check{T}$ and $\check{R}$. Define C_f, the *correct inputs* for f, to be the set of keys k for which $f(k) = f^*(k)$.

The adversary produces a correct output only if at least one of the n user keys

appears in C_f. For a given f, the probability that *none* of the keys appear in C_f is

$$\begin{aligned}\frac{\binom{p-|C_f|}{n}}{\binom{p}{n}} &> \frac{(p-|C_f|-n)^n}{p^n} \\ &= \left(1-\frac{|C_f|+n}{p}\right)^n \\ &> 1-\frac{n(|C_f|+n)}{p},\end{aligned}$$

and so the probability that *at least one* key appears in C_f is at most $\frac{n}{p}|C_f| + \frac{n^2}{p}$. Averaging over all f then gives

$$\Pr\left[f^*(k_i) = d_i \text{ for at least one } i\right] < \frac{n^2}{p} + \frac{n}{p^{n+1}} \sum_f |C_f|. \tag{4.5}$$

We will now use the bound on the number of distinct f^* to show that $\sum_f |C_f|$ is small.

Consider the set of all polynomials $f_1, f_2, \ldots f_m$ that map to a single function f^*, and their corresponding sets of correct keys $C_{f_1}, C_{f_2}, \ldots C_{f_m}$. Because any two degree $(n-1)$ polynomials are equal if they are equal on any n elements of Z_p, each n-element subset of Z_p can be contained in at most one of the C_{f_i}. On the other hand, each C_{f_i} contains exactly $\binom{|C_{f_i}|}{n}$ subsets of size n. Because there are only $\binom{p}{n}$ subsets of size n to partition between the C_{f_i}, we have

$$\sum_i \binom{|C_{f_i}|}{n} \leq \binom{p}{n},$$

and summing over all M choices of f^* then gives

$$\sum_f \binom{|C_f|}{n} \leq M\binom{p}{n}.$$

We now wish to bound the maximum possible value of $\sum_f |C_f|$ given this constraint.

Observe that $\binom{|C_f|}{n} > \frac{(|C_f|-n)^n}{n!}$ when $|C_f| \geq n$, from which it follows that

$$\sum_{f:|C_f|\geq n} (|C_f| - n)^n < n! \sum_f \binom{|C_f|}{n} < n! M \binom{p}{n}. \tag{4.6}$$

Now, $(|C_f| - n)^n$ is a convex function of $|C_f|$, so the left-hand side is minimized for fixed $\sum_f |C_f|$ by setting all $|C_f|$ equal. It follows that $\sum_f |C_f|$ is *maximized* for fixed $\sum_{f:|C_f|\geq n}(|C_f| - n)^n$ when all $|C_f|$ are equal.

Setting each $|C_f| = c$ and summing over all p^n values of f, we get

$$p^n(c-n)^n < n! M \binom{p}{n},$$

from which it follows that

$$c < \frac{1}{p}\left(n! M \binom{p}{n}\right)^{1/n} + n,$$

and thus that

$$\sum_f |C_f| \leq p^n c < p^{n-1}\left(n! M \binom{p}{n}\right)^{1/n} + np^n.$$

Plugging this bound back into (4.5) then gives

$$\begin{aligned}
\Pr_A[\vec{r} \neq 0^n] &= \Pr[f^*(k_i) = d_i \text{ for at least one } i] \\
&< \frac{2n^2}{p} + \frac{n}{p^2}\left(n! M \binom{p}{n}\right)^{1/n} \\
&< \frac{2n^2}{p} + \frac{n}{p^2}(Mp^n)^{1/n} \\
&= \frac{2n^2 + nM^{1/n}}{p}.
\end{aligned}$$

Using Claim 21, it is not hard to compute a limit on how much information the tamperer can remove before recovering any of the data becomes impossible:

Corollary 22 *Let (I, E, R) and $(\check{I}, \check{T}, \check{R})$ be as in Claim 21. Let $\epsilon > 0$ and let $p > 4n^3/\epsilon$. If for any fixed $\check{k}$, $\check{T}$ destroys at least $n \lg(n/\epsilon) + 1$ bits of entropy, then*

$$\Pr_A[\vec{r} = 0^n] \geq 1 - \epsilon.$$

Proof: Let $\epsilon' = \epsilon \,/\, \left(\frac{1}{2n} + 2^{-1/n}\right)$. If $\check{T}$ destroys at least $n \lg(n/\epsilon') + 1$ bits of entropy, then we have

$$M \leq p^n \cdot 2^{-(n \lg(n/\epsilon')+1)} = \frac{1}{2} p^n (n/\epsilon')^{-n} = \frac{1}{2} \left(p\epsilon'/n\right)^n . \tag{4.7}$$

Plug this into (4.4) to get:

$$\begin{aligned}
\Pr[\text{some } d_i \text{ is recovered}] &\leq \frac{2n^2 + nM^{1/n}}{p} \\
&\leq \frac{2n^2 + n\left(\frac{1}{2}(p\epsilon'/n)^n\right)^{1/n}}{p} \\
&= \frac{2n^2}{p} + 2^{-1/n}\epsilon' \\
&< \frac{2n^2}{4n^3/\epsilon'} + 2^{-1/n}\epsilon' \\
&= \epsilon'\left(\frac{1}{2n} + 2^{-1/n}\right) \\
&= \epsilon.
\end{aligned}$$

We thus have:

$$\Pr_A[\vec{r} = 0^n] = 1 - \Pr[\text{some } d_i \text{ is recovered}] \geq 1 - \epsilon.$$

	Destructive Tamperer	Arbitrary Tamperer
Standard Recovery	all-or-nothing	all-or-nothing
Public Recovery	all-or-nothing	symmetric recovery
Private Recovery	all-or-nothing	—

Table 4.1: Summary of results. "All-or-nothing" means that all-or-nothing integrity can be achieved in this model; "symmetric recovery" means that all-nothing integrity cannot be achieved, but symmetric recovery can; "—" means that no guarantees are possible.

4.6 Summary of the Study of Data Entanglement

Entangling documents of different users is a promising idea for strengthening the integrity of individual users' data, especially when the data is stored in an untrusted server. However, existing systems such as Dagster and Tangler only have an intuitive notion of entanglement that is insufficient by itself to provide much increased security. In this chapter, we analyzed the probability of destroying one document without affecting any other documents in these systems. Our analysis showed that the security they provide is not strong, even if we limit the class of attacks permitted against the entangled data.

Motivated by the desire to improve the security provided by entanglement, we defined the stronger notion of document dependency, in which destroying some document is guaranteed to destroy specific other documents, and all-or-nothing integrity, in which destroying some document is guaranteed to destroy all other documents. We considered a variety of potential attacks and showed for each what level of security was possible. These results are summarized in Table 4.6; they show that it is possible in principle to achieve all-or-nothing integrity with only mild restrictions on the adversary.

Whether it is possible in practice is a different question. Our model abstracts away most of the details of the storage and recovery processes, which hides unde-

sirable features of our algorithms such as the need to process all data being stored simultaneously and the need to read every bit of the data store to recover any data item. Some of these undesirable features could be removed with a more sophisticated model, such as a round-based model that treated data as arriving over time, allowing combining algorithms that would touch less of the data store for each storage or retrieval operation at the cost of making fewer documents depend on each other. The resulting system might look like a variant of Dagster or Tangler with stronger mechanisms for entanglement. But such a model might permit more dangerous attacks if the adversary is allowed to tamper with data during storage, and finding the right balance between providing useful guarantees and modeling realistic attacks will be necessary.

Chapter 5

Privacy-Preserving Data Mining for Association Rules

This chapter addresses a concrete scenario of computation with untrusted parties in the context of data mining for association rules. Standard algorithms for association-rule mining are based on identification of *frequent itemsets*. In this chapter, we consider how to maintain privacy in distributed mining of frequent itemsets [143]. That is, we study how two (or more) parties find frequent itemsets in a distributed database without revealing each party's portion of the data to the other. The previous solution for vertically partitioned data leaks a significant amount of information, while the previous solution for horizontally partitioned data only works for three parties or more. In this work, we apply the techniques from Chapter 2 to design algorithms for both vertically and horizontally partitioned data. We give two algorithms for vertically partitioned data; one of them reveals only the support count, and the other reveals nothing. Both of them have computational overheads linear in the number of transactions. Our algorithm for horizontally partitioned data works for two or more parties and is more efficient than the previous solution.

5.1 Background and Motivation

Data mining has been studied extensively and applied widely. Through the use of data-mining techniques, businesses can discover hidden patterns and rules from a database and then employ them to predict the future. An important case of data mining is *distributed data mining*, in which a database is distributed among two or more parties, and each party owns a portion of the data. These parties need to collaborate with each other so that they can jointly mine the data and produce results that are interesting to both of them. Privacy concerns are of great importance in this scenario, because each party does not want to reveal her own portion of the data, although she would like to participate in the mining.

This work is concerned with a major category of data mining, namely mining of *association rules*. Look at the transaction database of a supermarket. We may find that most of those who buy bread also buy milk. Therefore, "bread $\Rightarrow$ milk," which means "buying bread implies buying milk," is a candidate association rule. Two metrics are defined to measure such a candidate rule: *confidence* and *support*. Here confidence means the number of transactions in which both bread and milk are bought divided by the number of transactions in which bread is bought. Support means the number of transactions in which bread and milk are bought divided by the overall number of transactions. A candidate is considered a valid association rule if both its confidence and its support are sufficiently high.

Standard algorithms for association rule mining are based on identification of *frequent itemsets* [6]. We say that bread and milk constitute a frequent itemset if, in a sufficiently large percentage of transactions, both of them are bought (*i.e.* if its support is high). If all frequent itemsets can be computed, then all association rules can be computed easily from the frequent itemsets.

In this work, we study how to maintain privacy in distributed mining of frequent itemsets. That is, we study how two (or more) parties find frequent itemsets in a distributed database without revealing each party's portion of the data to the other. We will formally specify what we mean by "privacy." We will also give solutions for two major types of data partition, namely *vertical partition* and *horizontal partition*, and show that our algorithms preserve privacy.

Related Work To the best of our knowledge, Clifton and his students were the first to study privacy-preserving distributed mining of association rules/frequent itemsets. In [138], Vaidya and Clifton gave an algebraic solution for vertically partitioned data. However, this solution can *leak many linear combinations* of each party's private data to the other. Furthermore, to process one candidate frequent itemset, its computational overhead is quadratic in the number of transactions. In [86, 87], Kantarcioglu and Clifton gave a solution for horizontally partitioned data. However, this solution uses Yao's *generic* secure-computation protocol as a subprotocol; furthermore, it only works for three parties or more, not for two parties.

Privacy-preserving data mining has been a topic of active study (see, *e.g.*, papers by Agrawal and his collaborators [5, 4]). In particular, many papers have addressed the privacy issues in mining of association rules/frequent itemsets. Some examples are [39, 43, 122, 109, 128]. However, these papers are concerned with privacy of individual transactions and/or hiding of sensitive rules, rather than privacy in distributed mining.

Privacy-preserving distributed mining was first addressed by Lindell and Pinkas [90], but their paper only discusses the classification problem, not the association-rule problem.

As pointed out in [41], the problems of privacy-preserving data mining can be

viewed as an application of generic secure computation. Previous protocols for generic secure computation [142, 16, 61, 29] can solve such problems in theory. However, these generic protocols are highly expensive, and thus it is our goal to design special-purpose solutions that are much more efficient for our problems.

Our Contributions In this work, we rigorously specify the problems and privacy requirements of privacy-preserving mining of frequent itemsets. We give algorithms for vertically and horizontally partitioned data.

For vertically partitioned data, we design algorithms with two levels of privacy. The privacy guarantees for *both* levels are superior to those in previous works by others. Our algorithms are very efficient in that their computational overheads are linear in the number of transactions.

For horizontally partitioned data, our algorithm is more efficient than the previous solution. In addition, our algorithm works not only for three parties and above but also for two parties.

Chapter Organization The rest of this chapter is organized as follows. In Section 5.2, we present the problem formulation and the privacy requirements. In Sections 5.3 and 5.4, we describe two-party algorithms for vertically partitioned data, with weak privacy and strong privacy, respectively. In Section 5.5, we give a two-party algorithm for horizontally partitioned data. In Section 5.6, we show how to extend the algorithms to distributed mining with more than two parties. We summarize this work in Section 5.7.

5.2 Technical Preliminaries

5.2.1 Problem Formulation

Association Rules and Frequent Itemsets We adopt the following standard formulation of association-rule mining: Assume that $\mathcal{I} = \{I_1, \ldots, I_m\}$ is a set of literals, which are called *items*. We call any subset of $\mathcal{I}$ an *itemset*. Assume that $\mathcal{T} = \{T_1, \ldots, T_n\}$ is a set of transactions, where each transaction T_i is a set of items (*i.e.*, $T_i \subseteq \mathcal{I}$). We say that a transaction T_i *contains* an itemset X if and only if $X \subseteq T_i$. An *association rule* is of the form $X \Rightarrow Y$, where X and Y are non-empty itemsets such that $X \cap Y = \Phi$.

Such an association rule *holds* in the transaction set $\mathcal{T}$ with confidence $\alpha\%$ if $\alpha\%$ of the transactions containing X also contain Y. Such an association rule has support $\beta\%$ if $\beta\%$ of the transactions contain both X and Y.

The major technical problem in association-rule mining is *frequent itemset* identification. suppose that a sufficiently large β has been chosen. An itemset is frequent if and only if its support is greater than or equal to $\beta\%$.

Matrix Representation Mathematically, the transaction set $\mathcal{T}$ can be represented by a boolean matrix $\mathcal{D}$. Each row of the matrix corresponds to a transaction, while each column corresponds to an item. A matrix element $\mathcal{D}(i, j)$ is 1 if the ith transaction T_i contains the jth item I_j; it is 0 otherwise. The following example

illustrates how to convert the transaction set $\mathcal{T}$ to the boolean matrix $\mathcal{D}$.

	Bread	Milk	Eggs
Transaction 1	√		√
Transaction 2	√		
Transaction 3	√	√	√
Transaction 4		√	√

$$\Longrightarrow \quad \begin{matrix} 1 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{matrix}$$

We define the *support count* of an itemset as the number of transactions that contain this itemset. Formally, let C be the set of columns corresponding to an itemset. The support count of the itemset $\{I_j | j \in C\}$ is $S = |\{i | \forall j \in C,\ \mathcal{D}(i,j) = 1\}|$. Therefore, to decide whether the itemset $\{I_j | j \in C\}$ is frequent, we actually need to decide whether $S > \frac{\beta \cdot n}{100}$ (recall that n is the number of transactions).

As pointed out in [138], the support count S is essentially the inner product of all columns in set C. Because $\mathcal{D}$ is a *boolean* matrix,

$$\begin{aligned} S &= |\{i | \forall j \in C,\ \mathcal{D}(i,j) = 1\}| \\ &= |\{i | \prod_{j \in C} \mathcal{D}(i,j) = 1\}| \\ &= \sum_{i=1}^{n} \prod_{j \in C} \mathcal{D}(i,j) \\ (\ &= InnerProduct_{j \in C} \vec{\mathcal{D}_j}), \end{aligned}$$

where $\vec{\mathcal{D}_j} = (\mathcal{D}_{1,j}, \ldots, \mathcal{D}_{n,j})$ stands for the jth column of $\mathcal{D}$. Therefore, the problem of frequent itemset mining amounts to comparing this inner product with the threshold $\frac{\beta \cdot n}{100}$.

Vertical Partition and Horizontal Partition We consider our problem with respect to two types of data partition, namely vertical and horizontal. Intuitively, vertical partition means that each party owns some columns of the matrix $\mathcal{D}$, while horizontal partition means that each party owns some rows. For simplicity, at this point we only discuss two-party distributed mining and leave the extension to more parties to Section 5.6. Suppose that the two parties are A and B.

Formally, if the data are vertically partitioned, then A (resp., B) owns a set C_A (resp., C_B) of *columns* of the boolean matrix $\mathcal{D}$, where $C_A \cup C_B = [1, m]$ and $C_A \cap C_B = \Phi$. Recall that we are studying an itemset $\{I_j | j \in C\}$. The column set C of this itemset is partitioned into two subsets — $C \cap C_A$, which is owned by A, and $C \cap C_B$, which is owned by B. It is easy to see that

$$S = \sum_{i=1}^{n} \prod_{j \in C} \mathcal{D}(i,j) = \sum_{i=1}^{n} (\prod_{j \in C \cap C_A} \mathcal{D}(i,j) \cdot \prod_{j \in C \cap C_B} \mathcal{D}(i,j)).$$

Let $x_i = \prod_{j \in C \cap C_A} \mathcal{D}(i,j)$ and $y_i = \prod_{j \in C \cap C_B} \mathcal{D}(i,j)$. Note that A can privately compute all x_is and that B can privately compute all y_is. Let $t = \frac{\beta \cdot n}{100}$. Therefore, our problem can be formulated as follows.

Problem 23 *(Problem for Vertically Partitioned Data) A has a private input $\vec{x} = (x_1, \ldots, x_n)$ ($x_i \in \{0, 1\}$), and B has a private input $\vec{y} = (y_1, \ldots, y_n)$ ($y_i \in \{0, 1\}$). For a public threshold $t \in [0, n]$, design a two-party algorithm to decide whether $\sum_{i=1}^{n} x_i y_i > t$. This algorithm should be either* strongly *or* weakly privacy-preserving*, as define in Subsection 5.2.2.*

If the data are horizontally partitioned, then A (resp., B) owns a set R_A (resp., R_B) of *rows*, where $R_A \cup R_B = [1, n]$ and $R_A \cap R_B = \Phi$. It is easy to see

$$S = \sum_{i=1}^{n} \prod_{j \in C} \mathcal{D}(i,j) = \sum_{i \in R_A} \prod_{j \in C} \mathcal{D}(i,j) + \sum_{i \in R_B} \prod_{j \in C} \mathcal{D}(i,j).$$

Let $x = \sum_{i \in R_A} \prod_{j \in C} \mathcal{D}(i,j)$ and $y = \sum_{i \in R_B} \prod_{j \in C} \mathcal{D}(i,j)$. Note that A can privately compute x and B can privately compute y. Therefore, our problem can be formulated as follows.

Problem 24 *(Problem for Horizontally Partitioned Data) A has a private input $x \in [0,n]$, and B has a private input $y \in [0,n]$. For a public threshold $t \in [0,n]$, design a two-party algorithm to decide whether $x + y > t$. This algorithm should be either* strongly *or* weakly privacy-preserving*, as define in Subsection 5.2.2.*

5.2.2 Definitions of Privacy

As in previous works on privacy-preserving distributed mining [90, 138, 86, 87], we assume that the participants are semi-honest. We specify our privacy requirements by adapting the definition of security with respect to semi-honest parties in Chapter 2 to our distributed frequent-itemset mining problems. Our definitions will apply to both the problem for vertically partitioned data and the problem for horizontally partitioned data.

It is clear that, in the best possible case, we could have an algorithm that leaks nothing but its output. For frequent itemset mining, it is often also acceptable for an algorithm to leak the support count of a candidate. So we distinguish two levels of privacy, namely strong privacy and weak privacy.

Definition 25 *A two-party distributed algorithm for frequent-itemset mining is* strongly privacy-preserving *if it securely computes the output of Problem 23 or 24 with respect to semi-honest parties as definied in Definition 3.*

A two-party distributed algorithm for frequent itemset mining is weakly privacy-preserving *if it securely computes the* support count *with respect to semi-honest parties as definied in Definition 3 and then computes the output of Problem 23 or 24.*

5.3 A Weakly Privacy-Preserving Algorithm for Vertically Partitioned Data

5.3.1 Overview

Recall that, in the problem of vertically partitioned data, A has a private input $(x_1, \ldots, x_n)$ and B has a private input $(y_1, \ldots, y_n)$. We need to design an algorithm to decide whether $\sum_{i=1}^{n} x_i y_i > t$, where t is a public input.

We build our weakly privacy-preserving algorithm based on probabilistic public-key encryption. Consider a probabilistic public-key encryption scheme whose cleartext space is $\{0, 1\}$. Let $E_K(x_i, r_i)$ stand for an encryption of cleartext x_i using public key K and random string r_i. Let $D_k(Z_i)$ stand for the decryption of ciphertext Z_i using private key k. Assume that we have a rerandomization algorithm that can rerandomize any ciphertext in polynomial time. (One such encryption scheme was given by Goldwasser and Micali in [63].)

For weak privacy, we only need to compute $S = \sum_{i=1}^{n} x_i y_i$ securely and compare it to the threshold t. The main idea of our algorithm is that A counts the number of 1s in a random permutation of $(x_1 y_1, \ldots, x_n y_n)$ — this number is equal to $\sum_{i=1}^{n} x_i y_i$. As to privacy, A cannot learn more information, because she only sees a random permutation.

More specifically, the algorithm has 4 steps. In Step 1, A encrypts $(x_1, \ldots, x_n)$ using her *own* public key (so that B cannot decrypt them) and sends the encryptions to B. In Step 2, B computes encryptions of $(x_1 y_1, \ldots, x_n y_n)$ from these encryptions. Then B rerandomizes the newly computed encryptions, repermutes them, and sends them to A. In Step 3, A decrypts the encryptions she received, counts the number of 1s, and compare it to the threshold. In Step 4, B outputs the value it has received

from A.

The only thing left to explain is how B computes encryptions of $(x_1y_1, \ldots, x_ny_n)$ from encryptions of $(x_1, \ldots, x_n)$. Observe that $x_iy_i = x_i$ if $y_i = 1$, and $x_iy_i = 0$ otherwise. Therefore, if $y_i = 1$, B simply takes the encryption of x_i as an encryption of x_iy_i. Otherwise, B computes an encryption of 0.

5.3.2 Algorithm

Mine1(A, B, $\vec{x}$, $\vec{y}$)

A's Input: $\vec{x} = (x_1, \ldots, x_n)$ $(x_i \in \{0, 1\})$; (k, K);

B's Input: $\vec{y} = (y_1, \ldots, y_n)$ $(y_i \in \{0, 1\})$; K;

Public Input: $t \in [0, n]$.

Step 1

(1.1) For $i = 1, \ldots, n$, A encrypts x_i using public key K: $X_i = E_K(x_i, r_i)$, where r_i is picked uniformly at random.

(1.2) A sends $\vec{X} = (X_1, \ldots, X_n)$ to B.

Step 2

(2.1) For $i = 1, \ldots, n$, B computes Z_i, an encryption of $z_i = x_iy_i$ as follows:

- If $y_i = 1$, $Z_i = X_i$; otherwise, $Z_i = E_K(0, 0)$.

(2.2) For $i = 1, \ldots, n$, B rerandomizes Z_i.

(2.3) B repermutes $\vec{Z}$ as follows:

- For $i = 1, \ldots, n$, $Z_i = Z_{\pi(i)}$, where π is a random permutation on $[1, n]$.

(2.4) B sends $\vec{Z} = (Z_1, \ldots, Z_n)$ to A.

Step 3

(3.1) For $i = 1, \ldots, n$, A decrypts Z_i to get cleartext z_i: $z_i = D_k(Z_i)$.

(3.2) A counts the number of 1's in $\{z_1, \ldots, z_n\}$. If the the number is greater than t, then A outputs "This is a frequent itemset;" otherwise, A outputs "This is not a frequent itemset." A sends its output to B.

Step 4

B outputs the output it has received from A.

5.3.3 Security Analysis

Theorem 26 *Mine1 is weakly privacy-preserving, if the encryption scheme is semantically secure.*

Proof: We construct a simulator S_A as follows. First, S_A outputs A's input (which it also receives as input) and simulates the coin flips of A in the algorithm. Then, to simulate message $\vec{Z}$, S_A computes S encryptions of 1 and $n - S$ encryptions of 0, rerandomizes them at random and repermutes them at random.

We construct S_B as follows. First, S_B outputs B's input (which it also receives as input) and simulates the coin flips of B in the algorithm. Then S_B simulate message $\vec{X}$ by generating n random ciphertexts.

The computational indistinguishabilities between the views (for computing the support count) and the simulators' outputs immediately follow from the semantic security of the encryption scheme.

5.3.4 Efficiency Analysis

Computational Overhead The algorithm Mine1 needs to compute at most $2n$ encryptions, n rerandomizations, and n decryptions. How expensive these operations are depends on what encryption scheme we use. Assume that we use the Goldwasser-Micali encryption scheme. Then each encryption amounts to one modu-

lar multiplication, where the modulus is s bits long. Each rerandomization requires two modular multiplications. Decryption is more expensive — two modular exponentiations, which are equivalent to no more than $2s$ modular multiplications. To summarize, the overall computational overhead is no more than $(2s + 4)n$ modular multiplications.

Communication Overhead The algorithm Mine1 needs to transfer $2n$ items, each s bits long. Therefore, the overall communication overhead is $2sn$ bits.

5.4 A Strongly Privacy-Preserving Algorithm for Vertically Partitioned Data

5.4.1 Overview

Homomorphic Encryption We build our strongly privacy-preserving algorithm using homomorphic encryption. We need a probabilistic public-key encryption algorithm F that satisfies the following conditions:

- The cleartext space $\mathcal{M}$ is a large field of size $\Theta(2^s)$. In particular, the size is greater than $2n + 1$.
- It is *not* necessary to have an efficient decryption algorithm; however, there exists a polynomial-time algorithm that uses the private key to decide whether a ciphertext decrypts to 0.
- There is a polynomial-time rerandomization algorithm.

- F is additively homomorphic. That is, for $m_1, m_2 \in \mathcal{M}$,

$$F(m_1, r_1) \uplus F(m_2, r_2)$$

  is an encryption of $m_1 + m_2$, where $\uplus$ is an "addition" operation that can be performed without decrypting $F(m_1, r_1)$ or $F(m_2, r_2)$.

- F allows homomorphic computation of constant multiplication. That is, for $m_1 \in \mathcal{M}$ and constant c_1,

$$c_1 \circ F(m_1, r_1)$$

  is an encryption of $c_1 m_1$, where $\circ$ is a "constant multiplication" operation that can be performed without decrypting $F(m_1, r_1)$.

One example of F is a variant of ElGamal encryption: $F_K(m_i, r_i) = (g^{m_i}(K)^{r_i}, g^{r_i})$, where g is a generator of a group in which discrete logarithm is hard. The second condition above is satisfied, because we can use the private key to compute g^{m_i} from a ciphertext and compare it with g^0, *i.e.*, compare with 1. To satisfy the fourth condition, we define, for any ciphertexts (M_1, G_1) and (M_2, G_2),

$$(M_1, G_1) \uplus (M_2, G_2) = (M_1 M_2, G_1 G_2).$$

To satisfy the fifth condition, we define, for any constant c_1,

$$c_1 \circ (M_1, G_1) = (M_1^{c_1}, G_1^{c_1}).$$

The reader can easily verify that $\uplus$ implements addition and $\circ$ implements constant multiplication.

Algorithm Design Recall that the support count of the candidate itemset is S, *i.e.*, $S = \sum_{i=1}^{n} x_i y_i$. For strong privacy, our algorithm needs to decide whether $S > t$ without revealing S to either A or B. The main idea of our algorithm is that $S > t$ if and only if there exists a 0 in $(S-t-1, \ldots, S-t-n)$.[1] We would be able to solve this problem immediately if the vector $(S-t-1, \ldots, S-t-n)$ could be revealed to A or B. However, for strong privacy, this vector cannot be revealed. Therefore, we reveal a masked vector $(r_1(S-t-1), \ldots, r_n(S-t-n))$ instead, where $r_1, \ldots, r_n$ are random non-zero elements of $\mathcal{M}$. Note that this masked vector has a 0 if and only if the original vector has a 0. On the other hand, all non-zero elements in the original vector have been replaced by randomized elements in the masked vector, so that no extra information is leaked. In this way, the algorithm can decide whether $S > t$ without revealing any extra information.

More specifically, the algorithm has 4 steps. In Step 1, A encrypts $(x_1, \ldots, x_n)$ using her *own* public key (so that B cannot decrypt them) and sends the encryptions to B. In Step 2, B computes encryptions of $(r_1(S-t-1), \ldots, r_n(S-t-n))$ from these encryptions. Then B rerandomizes the newly computed encryptions, repermutes them, and sends them to A. In Step 3, A checks these encryptions to see whether there is one that decrypts to 0. In Step 4, B outputs the output it has received from A.

The only thing left to explain is how B computes encryptions of $(r_1(S-t-1), \ldots, r_n(S-t-n))$ from the encryptions of $(x_1, \ldots, x_n)$. Observe that, because $x_i, y_i \in \{0, 1\}$, we have $S = \sum_{i=1}^{n} x_i y_i = \sum_{y_i=1} x_i$. Therefore, B can sum up all encryptions of x_i where $y_i = 1$, to get an encryption of S. Then, using the homomorphic property of F, B can compute encryptions of $(r_1(S-t-1), \ldots, r_n(S-t-n))$,

[1] We can prove this fact as follows. $S > t \Leftrightarrow 0 < S - t \ (\leq n) \Leftrightarrow S - t \in [1, n] \Leftrightarrow$ there exists $i \in [1, n]$ such that $S - t = i \Leftrightarrow$ there exists $i \in [1, n]$ such that $S - t - i = 0 \Leftrightarrow$ there exists a 0 in $(S-t-1, \ldots, S-t-n)$.

because t is public and r_is are picked by herself.

5.4.2 Algorithm

Mine2(A, B, $\vec{x}$, $\vec{y}$)

A's Input: $\vec{x} = (x_1, \ldots, x_n)$ $(x_i \in \{0,1\})$; (k, K);

B's Input: $\vec{y} = (y_1, \ldots, y_n)$ $(y_i \in \{0,1\})$; K;

Public Input: $t \in [0, n]$.

Step 1

(1.1) For $i = 1, \ldots, n$, A encrypts x_i using public key K: $X_i = F_K(x_i, r_i)$, where r_i is picked uniformly at random.

(1.2) A sends $\vec{X} = (X_1, \ldots, X_n)$ to B.

Step 2

(2.1) B computes an encryption $\overline{S}$ of $S = \sum_{i=1}^{n} x_i y_i$ as follows:

- $\overline{S} = F_K(0, 0)$;
- For $i = 1, \ldots, n$, if $y_i = 1$, $\overline{S} = \overline{S} \uplus X_i$.

(2.2) For $i = 1, \ldots, n$, B picks $r_i \in \mathcal{M} - \{0\}$ uniformly at random and computes an encryption of $r_i(S - t - i)$:

$$U_i = r_i \circ (\overline{S} \uplus F(-t - i, 0)).$$

(2.3) For $i = 1, \ldots, n$, B rerandomizes U_i.

(2.4) B repermutes $\vec{U} = (U_1, \ldots, U_n)$ as follows:

- For $i = 1, \ldots, n$, $U_i = U_{\pi(i)}$, where π is a random permutation on $[1, n]$.

(2.5) B sends $\vec{U} = (U_1, \ldots, U_n)$ to A.

Step 3

If one of U_is decrypts to 0, A outputs "This is a frequent itemset;" otherwise, A outputs "This is not a frequent itemset." A also sends its output to B.

Step 4

B outputs the output it has received from A.

5.4.3 Security Analysis

Theorem 27 *Mine2 is strongly privacy-preserving if the encryption scheme is semantically secure.*

Proof: We construct a simulator S_A as follows. S_A outputs the input of A (which it also receives as input) and simulates the coin flips of A in the algorithm. If this is a frequent itemset, S_A simulates message $\vec{U}$ using one random encryption of 0 and $n-1$ random encryptions of random elements of $\mathcal{M}-\{0\}$, in a random order; otherwise, it simulates message $\vec{U}$ using n random encryptions of random elements of $\mathcal{M}-\{0\}$, in a random order.

We construct S_B as follows. S_B outputs the input of B (which it also receives as input) and simulates the coin flips of B in the algorithm. S_B simulates message $\vec{X}$ using n random ciphertexts.

The computational indistinguishability of the views and the simulators' outputs immediately follows from the semantic security of the encryption scheme.

5.4.4 Efficiency Analysis

Computational Overhead The algorithm Mine2 needs to compute $2n+1$ encryptions and n rerandomizations. It also needs to check n ciphertexts to see whether they decrypt to 0. In addition, it needs to compute $\uplus$ a total of $2n$ times and $\circ$ a

total of n times. Therefore, its computational overhead is still linear in n and much lower than the previous solution.

Assume that we use the variant of ElGamal encryption: $F_K(m_i, r_i) = (g^{m_i}(K)^{r_i}, g^{r_i})$. Then each encryption amounts to three modular exponentiations plus one modular multiplication, where the modulus is s-bit. Each rerandomization is worth two modular exponentiations plus two modular multiplications. It takes one modular exponentiation plus one modular multiplication to check whether a ciphertext decrypts to 0. Computing $\uplus$ costs two modular multiplications, while computing $\circ$ costs two modular exponentiations. To summarize, the overall computational overhead is no more than $(11s + 9)n + (3s + 1)$ modular multiplications.

Communication Overhead The communication overhead of Mine2 is also $2sn$ bits.

5.5 An Algorithm for Horizontally Partitioned Data

5.5.1 Overview

Recall that, in the problem for horizontally partitioned data, A has a private input x and B has a private input y. We need to design an algorithm to decide whether $x + y > t$ where t is public.

We still build a strongly privacy-preserving algorithm based on homomorphic encryption. We use the homomorphic encryption scheme F specified in Section 5.4.

For strong privacy, our algorithm needs to decide whether $x + y > t$ without revealing x to B or revealing y to A. The main idea of our algorithm is that $x+y > t$ if and only if there exists a 0 in $(x + y - t - 1, \ldots, x + y - t - n)$ (see the footnote in Subsection 5.4.1 to see why this is true). However, for strong privacy, the vector

$(x+y-t-1,\ldots,x+y-t-n)$ cannot be revealed to A or B. Therefore, we reveal a masked vector $(r_1(x+y-t-1),\ldots,r_n(x+y-t-n))$ instead, where $r_1,\ldots,r_n$ are random non-zero elements of $\mathcal{M}$. Note that this masked vector has a 0 if and only if the original vector has a 0. On the other hand, all non-zero elements in the original vector have been replaced by randomized elements in the masked vector. In this way, the algorithm can decide whether $x+y>t$ without revealing any extra information.

More specifically, the algorithm has 4 steps. In Step 1, A encrypts x using her *own* public key (so that B cannot decrypt them) and sends the encryptions to B. In Step 2, B computes encryptions of $(r_1(x+y-t-1),\ldots,r_n(x+y-t-n))$ from the encryption of x, using the homomorphic property of F. Then B rerandomizes the newly computed encryptions, repermutes them, and sends them to A. In Step 3, A checks these encryptions to see whether there is one that decrypts to 0. In Step 4, B outputs the output it has received from A.

5.5.2 Algorithm

Mine3(A, B, x, y)

A's Input: $x \in [0,n]$; (k,K);

B's Input: $y \in [0,n]$; K;

Public Input: $t \in [0,n]$.

Step 1

(1.1) A encrypts x: $X = F_K(x,r)$, where r is picked uniformly at random.

(1.2) A sends X to B.

Step 2

(2.1) For $i=1,\ldots,n$, B picks $r_i \in \mathcal{M}-\{0\}$ and computes U_i, an encryption of

$u_i = r_i(x + y - t - i)$:

$$U_i = r_i \circ (X \uplus F_K(y - t - i, 0)).$$

(2.2) For $i = 1, \ldots, n$, B rerandomizes U_i.

(2.3) B repermutes $\vec{U} = (U_1, \ldots, U_n)$ at random.

(2.4) B sends $\vec{U} = (U_1, \ldots, U_n)$ to A.

Step 3

If one of U_is decrypts to 0, A outputs "This is a frequent itemset;" otherwise, A outputs "This is not a frequent itemset." A also sends the output to B.

Step 4

B outputs the output it has received from A.

5.5.3 Security Analysis

Theorem 28 *Mine3 is strongly privacy-preserving, if the encryption scheme is semantically secure.*

Proof: We construct a simulator S_A as follows. S_A outputs the input of A (which it also receives as input) and simulates the coin flips of A in the algorithm. If this is a frequent itemset, S_A simulates message $\vec{U}$ using one random encryption of 0 and $n-1$ random encryptions of random elements of $\mathcal{M} - \{0\}$, in a random order; otherwise, S_A simulates $\vec{U}$ using n random encryptions of random elements of $\mathcal{M} - \{0\}$, in a random order.

We construct S_B as follows. S_B outputs the input of B (which it also receives as input) and simulates the coin flips of B in the algorithm. S_B simulates message X using a random encryption.

The computational indistinguishability of the views and the simulators' outputs

immediately follows from the semantic security of the encryption scheme.

5.5.4 Efficiency Analysis

Computational Overhead The algorithm Mine3 needs to compute $n+1$ encryptions and n rerandomizations. It also needs to check n ciphertexts to see whether they decrypt to 0. In addition, it needs to compute $\uplus$ and $\circ$ each n times.

Assume that we use the variant of the ElGamal encryption scheme: $F_K(m_i, r_i) = (g^{m_i}(K)^{r_i}, g^{r_i})$. Then the overall computational overhead is $(8s+6)n + (3s+1)$ modular multiplications.

The previous solution for horizontally partitioned data only works for three parties or more. In Section 5.6, we will show how to extend our algorithm Mine3 to more parties.

Communication Overhead The communication overhead of Mine3 is $sn+s$ bits.

5.6 Extension to Multi-party Distributed Mining

In this section, we demonstrate how to extend our algorithms to multi-party distributed mining. To avoid overly complicated notations, instead of presenting general algorithms for k parties, we give three-party algorithms for vertically and horizontally partitioned data. It is straightforward to further extend our algorithms to more parties in a similar way.

5.6.1 An Algorithm for Vertically Partitioned Data

Now we extend our strongly privacy-preserving algorithm Mine2 to three-party distributed mining.

Suppose that we have the third party C with private input $(z_1, \ldots, z_n)$. The extended algorithm has 5 steps. In Step 1, A encrypts $(x_1, \ldots, x_n)$ using her own public key and sends the encryptions to B. In Step 2, B computes encryptions of $(x_1y_1, \ldots, x_ny_n)$ and sends them to C. In Step 3, C computes $(r_1(\sum_{i=1}^{n} x_iy_iz_i - t - 1), \ldots, r_n(\sum_{i=1}^{n} x_iy_iz_i - t - n))$. Then C rerandomizes these newly computed encryptions, repermutes them, and sends them to A. In Step 4, A checks the encryptions she received to see whether there is one that decrypts to 0. In Step 5, B and C output the output they have received from A. Note that Steps 1, 3, 4, and 5 of the extended algorithm correspond to Steps 1, 2, 3, and 4 of algorithm Mine2, respectively. The only new step is Step 2, which is based on the fact that $x_iy_i = x_i$ if $y_i = 1$, and $x_iy_i = 0$ otherwise.

Mine4(A, B, C, $\vec{x}$, $\vec{y}$, $\vec{z}$)

A's Input: $\vec{x} = (x_1, \ldots, x_n)$ $(x_i \in \{0, 1\})$; (k, K);

B's Input: $\vec{y} = (y_1, \ldots, y_n)$ $(y_i \in \{0, 1\})$; K;

C's Input: $\vec{z} = (z_1, \ldots, z_n)$ $(z_i \in \{0, 1\})$; K;

Public Input: $t \in [0, n]$.

Step 1

(1.1) For $i = 1, \ldots, n$, A encrypts x_i using public key K: $X_i = F_K(x_i, r_i)$, where r_i is picked uniformly at random.

(1.2) A sends $\vec{X} = (X_1, \ldots, X_n)$ to B.

Step 2

(2.1) For $i = 1, \ldots, n$, B computes U_i, an encryption of $u_i = x_iy_i$ as follows:

- If $y_i = 1$, B sets $U_i = X_i$; otherwise, B sets $U_i = F_K(0, 0)$;
- B rerandomizes U_i.

(2.2) B sends $\vec{U} = (U_1, \ldots, U_n)$ to C.

Step 3

(3.1) C computes an encryption of $v = \sum_{i=1}^{n} x_i y_i z_i$ as follows:

- $V = F_K(0, 0)$;
- For $i = 1, \ldots, n$, if $z_i = 1$, $V = V \uplus U_i$.

(3.2) For $i = 1, \ldots, n$, C picks $r_i \in \mathcal{M} - \{0\}$ uniformly at random and computes an encryption of $r_i(v - t - i)$:

$$W_i = r_i \circ (V \uplus F_K(-t - i, 0)).$$

(3.3) For $i = 1, \ldots, n$, C rerandomizes W_i.

(3.4) C repermutes $\vec{W} = (W_1, \ldots, W_n)$ at random.

(3.5) C sends $\vec{W} = (W_1, \ldots, W_n)$ to A.

Step 4

If one of W_is decrypts to 0, A outputs "This is a frequent itemset;" otherwise, A outputs "This is not a frequent itemset." A also sends the output to B and C.

Step 5

B and C output the output they have received from A.

5.6.2 An Algorithm for Horizontally Partitioned Data

Now we extend our algorithm Mine3 to three-party distributed mining.

Suppose that we have the third party C with private input z. The extended algorithm has 5 steps. In Step 1, A encrypts x using her own public key and sends it to B. In Step 2, B computes an encryption of $x + y$ and sends it to C. In Step 3, C computes encryptions of $(r_1(x+y+z-t-1), \ldots, r_n(x+y+z-t-n))$ using the homomorphic property of F. Then C rerandomizes these newly computed encryptions,

repermutes them, and sends them to A. In Step 4, A checks the encryptions she received to see whether there is one that decrypts to 0. In Step 5, B and C output the output they have received from A. Note that Steps 1, 3, 4, and 5 of the extended algorithm correspond to Steps 1, 2, 3, and 4 of algorithm Mine3, respectively. The only new step is Step 2, which is also based on the homomorphic property of F.

Mine5(A, B, C, x, y, z)

A's Input: $x \in [0, n]$; (k, K);

B's Input: $y \in [0, n]$; K;

C's Input: $z \in [0, n]$; K;

Public Input: $t \in [0, n]$.

Step 1

(1.1) A encrypts x: $X = F_K(x, r)$, where r is picked uniformly at random.

(1.2) A sends X to B.

Step 2

(2.1) B computes U, an encryption of $u = x + y$: $U = X \uplus F_K(y, 0)$.

(2.2) B rerandomizes U.

(2.3) B sends U to C.

Step 3

(3.1) For $i = 1, \ldots, n$, C picks $r_i \in \mathcal{M} - \{0\}$ and computes V_i, an encryption of $v_i = r_i(x + y + z - t - i) = r_i(u + z - t - i)$:

$$V_i = r_i \circ (U \uplus F_K(z - t - i, 0)).$$

(3.2) For $i = 1, \ldots, n$, C rerandomizes V_i.

(3.3) C repermutes $\vec{V} = (V_1, \ldots, V_n)$ at random.

(3.4) C sends $\vec{V} = (V_1, \ldots, V_n)$ to A.

Step 4

If one of V_is decrypts to 0, A outputs "This is a frequent itemset;" otherwise, A outputs "This is not a frequent itemset." A also sends the output to B and C.

Step 5

B and C output the output they have received from A.

5.7 Summary of the Work on Data Mining

In this chapter, we study privacy-preserving algorithms for distributed mining of frequent itemsets. Our algorithms provide very strong privacy guarantee as defined in cryptography. They have computational overheads linear in the number of transactions and therefore are very efficient.

The work described in this chapter is the most direct application of the techniques presented in Chapter 2. There are other ways to apply those techniques to the problems addressed in this chapter. We choose what we have presented because it is the most efficient option.

Chapter 6

Secure Mobile-Agent Computation

Yet another concrete scenario of computation with untrusted parties is mobile agent computation. The major security concern here is privacy. In this joint work with Yang Richard Yang [145], I study the security issues in mobile-agent computation. It is known that oblivious transfer (OT) from a trusted party can be used to protect the agent's privacy and the hosts' privacy.[1] In this work, we introduce a new cryptographic primitive called *Verifiable Distributed Oblivious Transfer (VDOT)*, which allows us to replace a single trusted party with a group of threshold trusted servers. The design of VDOT uses a novel technique *consistency verification of encrypted secret shares* to protect the privacy of both the sender and the receiver against malicious attacks of the servers. We also show the design of a system to apply VDOT to protect the privacy of mobile agents. Our design partitions an agent into the general portion and the security-sensitive portion. We implement the key components of our system. Our preliminary evaluation shows that protecting mobile agents not only is possible, but also can be implemented efficiently.

[1]Note that this is *not* the only way to protect privacy for mobile agents. For example, X-Cash, which was proposed by Jakobsson and Juels, can also be used to protect privacy in e-commerce generally.

6.1 Background and Motivation

As an important paradigm of computation, the mobile agent has the a lot of potential applications in electronic commerce. However, the success of the mobile agents depends on security. In the past, the focus of mobile-agent security has been on protecting the safety and the integrity of visited hosts. To achieve this objective, researchers have proposed novel techniques such as the Sandbox architecture [67], which restricts the access of a visiting mobile agent, and proof-carrying code [103], which allows a host to efficiently verify that the visiting mobile agent will not do harm to the host.

However, in mobile agent computing, it is as important to protect the privacy of the agent from the hosts as to protect the privacy of the hosts from the agent. Since Sander and Tschudin's pioneering work [125], various systems have been designed for this purpose [126, 25, 8]. In particular, Algesheimer, Cachin, Camenisch, and Karjoth [8] present a nice and general solution that has provable security. However, the security of this system relies on a single trusted party which carries out oblivious transfer (OT). If the trusted party is compromised, the privacy of both the agent and the hosts can be violated.

The security of [8] can be significantly strengthened if the single trusted party is replaced by a group of threshold trusted servers. For this end, a "threshold extension" of OT is needed. One possible solution is to use Naor and Pinkas's distributed OT (DOT) [102], which involves a sender, a receiver, and a group of servers. In DOT, the sender has two items and the receiver chooses to receive one of them. First, the sender distributes to each server some data derived from her items, in such a secure way that no single server can figure out any information about her items. Then the receiver queries the servers. From the servers' responses, the receiver is able to

reconstruct one and only one of the two items. Furthermore, the receiver has no information about the other item and the sender has no information about which of the items the receiver has chosen.

However, DOT assumes semi-honest servers. If some servers are malicious, they can mislead the receiver to reconstruct a false item. To deal with such malicious servers, we propose a new cryptographic primitive called "Verifiable Distributed Oblivious Transfer," or VDOT for short.

Challenges and Contributions The design of VDOT is technically challenging. One might suggest that the objective of VDOT could be achieved by a secret-sharing scheme with oblivious transfer of each share. However, there are two somewhat conflicting goals that need to be achieved. On the one hand, the receiver must be able to verify the correctness of *both* items; otherwise, a malicious server could violate the receiver's privacy by tampering with its share of one item and observing whether or not this attack is detected by the receiver. On the other hand, in order to protect the sender's privacy, the receiver should be able to reconstruct *only one* of the two items. In summary, the major technical challenge is to allow the receiver to reconstruct only one item but verify the correctness of both items.

Our VDOT protocol uses a novel technique to address the above challenge. An overview of the VDOT protocol is as follows. During initialization, a global private key is shared in the Feldman VSS. An advantage of this setup is that the consistency of secret shares encrypted using ElGamal can be verified. Before each transfer, the sender distributes the shares of both items among the servers. During the transfer procedure, the receiver invokes the one-round OT protocol by Bellare and Micali [14, 25], with each server in a quorum (called *main servers*) in order to get the share of the item he chooses. Although the receiver can reconstruct only one item, he

can verify the consistency of both items through the help of the remaining servers (called *verification servers*), because the *encryptions of* the shares of both items are transfered to the receiver during the OT.

We then apply VDOT to mobile agent security to implement the key components of a mobile agent architecture. As far as we know, this is the first effort to implement a system that protects the privacy of mobile agents. To write an agent in our system, the designer extracts the security-sensitive portion of the agent into a function. Then the function is encoded as a garbled circuit, which is carried by the agent. Because we only apply the security mechanism to the security-sensitive portion of an agent, our system is efficient. Because the result of the security-sensitive portion is interpreted by the normal portion of the agent, all that a host needs to provide is an interpreter of garbled circuits. As a result, our system provides a general-purpose solution. We measure the overhead of our system and show that the overhead is acceptable. In other words, our preliminary evaluation shows that protecting mobile agents not only is possible, but also can be implemented efficiently.

In summary, the contributions of this work are as follows. First, we introduce a new cryptographic primitive, VDOT, which can be used in situations where proxies of OT are needed but no single proxy can be trusted. In particular, VDOT can be used to strengthen the security of the mobile agent system designed in [8]. Second, the design of VDOT uses a novel technique to achieve consistency verification of encrypted secret shares. Third, we apply VDOT to the problem of mobile agent security to implement the key components of an architecture for mobile agents.

Chapter Organization The rest of this chapter is organized as follows. In Subsection 6.1.1, we discuss related work. In Section 6.2, we define the security requirements for VDOT. (In principle, we can use the general definitions of secure multi-party

computation with respect to malicious adversaries in [59]; however, these general definitions are much more complicated than the definitions we give specifically for VDOT.) In Section 6.3, we present how VDOT implements consistency verification of encrypted shares. We prove the security properties of VDOT in Section 6.4. In Section 6.5, we show how to apply VDOT to a mobile-agent system. In Section 6.6, we present implementation issues and report initial performance evaluation. We summarize this work in Section 6.7.

6.1.1 Related Work

OT Protocols Oblivious Transfer was first introduced by Rabin [119]. Later, several variations were proposed, *e.g.*, 1-out-of-2 OT [42], 1-out-of-N OT [20], k-out-of-N OT [100], and adaptive k-out-of-n OT [101]. Our work can be viewed as an extension of DOT [102], which introduces a group of servers to the 1-out-of-2 OT scenario. The major difference is that, as we have explained, our VDOT protocol considers potentially malicious servers, while the DOT protocol considers semi-honest servers. Another difference is that our model allows the receiver to communicate with all servers.

PIR/SPIR Protocols A problem similar to OT is private information retrieval (PIR) [31], in which a user (analogous to the receiver in OT) privately retrieves a bit from a database (analogous to the sender in OT). However, in PIR, only the user's privacy is protected, and the amount of communication is required to be small. In order to get nontrivial solutions with information-theoretic privacy, it is often assumed that there are two or more copies of the database, held by database servers that do not communicate with each other. With computational assumptions, a PIR protocol with a single copy of the database can be constructed [89]. Gertner, Ishai,

Kushilevitz, and Malkin added the privacy of the database to the PIR model [55]. The result is called symmetric PIR (SPIR). The difference between SPIR and OT is that the former further requires small-communication overhead.

Interestingly, Gertner, Goldwasser, and Malkin introduced auxiliary servers to PIR [54], just as Naor and Pinkas introduced a group of servers to OT. However, in the Gertner-Goldwasser-Malkin model, the database itself is still involved in the protocol after the initialization stage, and the auxiliary servers may contain no information about the data at all (in the case of "total independence"). Therefore, it is significantly different from the models of distributed OT and verifiable distributed OT. The relationship between OT and PIR/SPIR is further studied in [36].

6.2 VDOT Definitions

We formulate the problem of VDOT as follows. A VDOT protocol involves a sender, a receiver, and a group of servers, $T_1, \ldots, T_n$. Each of the honest parties is a probabilistic Turing machine who is restricted to run in time polynomial in a security parameter s, while all the dishonest parties are controlled by an adversary who is also a probabilistic Turing machine running in time polynomial in s. We assume an authenticated, untappable channel between the sender (resp., receiver) and each server. Let $x_0, x_1 \in \{0,1\}$ be the two items held privately by the sender. Let $\sigma \in \{0,1\}$ be a private input of the receiver.

A VDOT protocol consists of an initialization stage and a transfer stage.[2] In the initialization stage, the sender sends a function $F_j : \{0,1\}^* \to \{0,1\}^*$ to each server T_j, where F_j depends on (x_0, x_1) and the sender's coin tosses. In the transfer

[2]We assume that all participants of the protocol, including the malicious ones, will proceed to the end of the protocol. Therefore, there is no *fairness* problem. This is a reasonable assumption because we *detect* cheating when any participant aborts the protocol, and in many realistic scenarios it is good enough to detect cheating.

stage, in order to learn x_σ, the receiver sends query q_j to server T_j and receiving reply $r_j = F_j(q_j)$ from T_j. Because the receiver may not send his queries all at once, q_j may depend on the replies to previous queries. After receiving replies from the servers, the receiver decides either to accept the replies (and gives an output $O(r_1, \ldots, r_n)$ which is supposed to be x_σ) or to reject the replies (and output $\perp$ which means cheating is detected).

We summarize the security requirements of a VDOT protocol as follows.

Definition 29 *(*correctness*) A VDOT protocol is correct if the receiver's outputs x_σ when all parties follow the protocol.*

Definition 30 *(*receiver's privacy*) A VDOT protocol protects the receiver's privacy against a coalition of the sender and t_1 servers if, for σ chosen uniformly at random from $\{0,1\}$, for any probabilistic polynomial-time adversary that controls a colluding group of the sender and t_1 servers, when all parties out of the colluding group are honest, the probability that the adversary outputs σ is at most $\frac{1}{2} + \epsilon$, where ϵ is negligible in s.*

Definition 31 *(*sender's privacy*) A VDOT protocol protects the sender's privacy against a coalition of the receiver and t_2 servers if, for (x_0, x_1) chosen uniformly at random from $\{0,1\}^2$, for any probabilistic polynomial-time adversary that controls a colluding group of the receiver and t_2 servers, for any random tape the adversary uses, when all parties out of the colluding group are honest, there exists $\sigma' \in \{0,1\}$ such that the probability that the adversary outputs $x_{1-\sigma'}$ is at most $\frac{1}{2} + \epsilon$, where ϵ is negligible in s.*

For verifiability, we require that cheating be detected if it may lead the receiver to compute a false x_σ. On the other hand, if the cheating behavior of some servers does not affect correct reconstruction of x_σ, it will be unnecessary to detect it.

Definition 32 *(*verifiability of reconstruction*) A VDOT protocol is verifiable if, when the sender and the receiver are honest, there exists a probabilistic polynomial-time algorithm V such that*

- $V(r_1, \ldots, r_n) =$ *"accept" if no server cheats;*
- $V(r_1, \ldots, r_n) =$ *"reject" with high probability if* $O(r_1, \ldots, r_n) \neq x_\sigma$.

Remark: In the above definition, we do not have any requirement of V's output if some server is cheating but $O(r_1, \ldots, r_n) = x_\sigma$. In this case, both acceptance and rejection will be fine, because there is cheating but it does not affect the reconstruction.

6.3 A VDOT Protocol

In this section, we address the technical challenge mentioned in Section 6.1 and present our protocol. Before describing our protocol in details, we first review an adapted version of the Bellare-Micali OT protocol, which can be understood as transferring both items encrypted using ElGamal. Then we show how to verify the consistency of secret shares encrypted using ElGamal, which is the key technical contribution of our protocol.

Recall that p, q are large primes such that $p = 2q + 1$, that G_q is the quadratic residue subgroup of Z_p^*, and that g is a generator of G_q. We assume that q has s bits.

6.3.1 Bellare-Micali OT

Assume that there exists a public random source. In this adapted version of Bellare-Micali OT, the receiver first picks $\delta \in G_q$ using the public random source. Because

the receiver has no control over the public random source, he does not know $\log_g \delta$, the discrete logarithm of δ. The receiver then picks $\beta \in [0, q-1]$ and sets

$$G_\sigma = g^\beta, \qquad\qquad G_{1-\sigma} = \delta / g^\beta.$$

Note that the receiver knows $\log_g G_\sigma$ but not $\log_g G_{1-\sigma}$. The receiver sends G_0, G_1, δ to the sender, along with a proof that he knows one of the two discrete logarithms, $\log_g G_0$ and $\log_g G_1$, using a result by Cramer *et al.* [33]. The sender first verifies that δ has been chosen properly according to the public random source, and $\delta = G_0 G_1$. Then the sender computes, for $b = 0, 1$,

$$\hat{x}_b = x_b G_b^k,$$

where $k \in [0, q-1]$ is the sender's private key and $K = g^k$ her public key.

This OT protocol can be understood as transferring both items in ElGamal ciphertexts. Recall that in the ElGamal encryption scheme, which is semantically secure under the DDH assumption, when cleartext $m \in G_q$ is encrypted with public key K using random string $r \in [0, q-1]$, the ciphertext will be $(mK^r, g^r) = (m(g^r)^k, g^r)$. In the Bellare-Micali OT above, $\hat{x}_b$ can be understood as the first element of the ElGamal ciphertext of x_b, encrypted using random string $\log_g G_b$. For convenience, hereafter we often refer to the first element of an ElGamal ciphertext as *the ciphertext*. In order to decrypt $\hat{x}_b$, a party not knowing k (*e.g.*, the receiver) must know $\log_g G_b$, the random string used for encryption.

The sender gives both $\hat{x}_0$ and $\hat{x}_1$ to the receiver. Because $G_\sigma^k = (g^\beta)^k = K^\beta$, the receiver can reconstruct x_σ by computing

$$x_\sigma = \hat{x}_\sigma / K^\beta,$$

where K is public and β is known to the receiver. However, because the receiver does not know $\log_g G_{1-\sigma}$, he cannot compute $s_{1-\sigma}$.

6.3.2 Consistency Verification

The basis of our VDOT protocol is actually a distributed version of the above Bellare-Micali OT. The sender distributes shares of the two items, x_0 and x_1 respectively, among the servers; then each server runs the above Bellare-Micali OT with the receiver, to transfer the shares of the two items, such that the shares of x_σ, but not $x_{1-\sigma}$, can be received by the receiver. The privacy properties of our protocol are based on the privacy properties of Bellare-Micali OT. Therefore, our remaining question is how the receiver detects cheating if any server does not transfer the correct share.

To detect cheating, the receiver can verify the *consistency* of shares. More precisely, suppose that $(s_1, \ldots, s_n)$ are the shares of a secret using (n, t)-Shamir secret sharing. Then, for any quorum J ($|J|$=t), any $i \notin J$, it must hold that

$$\sum_{j \in J} s_j \cdot \prod_{l \in J}^{l \neq j} \frac{i-l}{j-l} = s_i.$$

Now suppose that we consider a variant of Shamir scheme by applying a homomorphic mapping $\alpha \rightarrow g^\alpha$ to the Shamir scheme. Then, for secret shares $(s_1, \ldots, s_n)$, it must hold that

$$\prod_{j \in J} s_j^{\prod_{l \in J}^{l \neq j} \frac{i-l}{j-l}} = s_i.$$

We say that s_i is consistent with $\{s_j | j \in J\}$ whenever the above equation holds. Therefore, if no share is corrupted, any share should be consistent with any disjoint quorum. But if some shares are corrupted while others are not, with high probability there is inconsistency that can be detected.

However, note that the receiver needs to detect inconsistency of shares of *either* item. For $x_{1-\sigma}$, the receiver only sees the encryptions of its shares, but not its shares in cleartext. To allow the receiver to detect inconsistency on encrypted shares, we need to use a property of Feldman VSS.

Specifically, suppose that the servers share k using (n,t)-Feldman VSS. Denote by k_j the share of k held by T_j, and $K_j = g^{k_j}$ the corresponding committment. Then clearly,

$$\prod_{j\in J} K_j^{\prod_{l\in J}^{l\neq j} \frac{i-l}{j-l}} = K_i.$$

Therefore, for the shares $(s_1,\ldots,s_n)$ in the above variant of Shamir scheme,

$$\begin{aligned} \prod_{j\in J} s_j^{\prod_{l\in J}^{l\neq j} \frac{i-l}{j-l}} = s_i \qquad &\Leftrightarrow \qquad \prod_{j\in J} (s_j K^{\beta})^{\prod_{l\in J}^{l\neq j} \frac{i-l}{j-l}} = (s_i K^{\beta}) \\ &\Leftrightarrow \qquad \prod_{j\in J} (s_j K^{1-\beta})^{\prod_{l\in J}^{l\neq j} \frac{i-l}{j-l}} = (s_i K^{1-\beta}). \end{aligned}$$

The above means that, to verify consistency among shares (the equation on the left side), the receiver only needs to check an identity on the right side, which only involves encrypted shares (which the receiver is able to see).

6.3.3 A VDOT Protocol Specification

In this subsection, we give the full details of our protocol. A server T_j is called "main server" if $1 \leq j \leq t$; it is called "verification server" otherwise.

System Initialization An (n,t)-Feldman VSS is set up among $T_1,\ldots,T_n$ to share k, the sender's private key.

Step 0: The sender distributes the shares of x_0 and x_1 (in the variant of Shamir's scheme with threshold t), respectively, among $T_1,\ldots,T_n$.

Step 1: The receiver picks $\delta \in G_q$ uniformly at random according to the public random source. He also picks $\beta \in [0, q-1]$ uniformly at random, and computes $G_\sigma = g^\beta$ and $G_{1-\sigma} = \delta / g^\beta$. He sends query (G_0, G_1, δ) to each main server, along with a proof that he knows one of the two discrete logarithms, $\log_g G_0$ and $\log_g G_1$.

Step 2: Each main server T_j first verifies that 1) the receiver's proof is valid; 2) δ is chosen properly according to the public random source; and 3) $\delta = G_0 G_1$ $(G_0, G_1, \delta \in G_q)$. If all the three conditions are satisfied, T_j computes, for $b = 0, 1$,

$$\hat{s}_{b,j} = s_{b,j} G_b^{\kappa_j}, \tag{6.1}$$

where $s_{b,j}$ is T_j's share of x_b in the variant of Shamir scheme. T_j sends $(\hat{s}_{0,j}, \hat{s}_{1,j})$ to the receiver.

Step 3: The receiver checks, for each j, that $\hat{s}_{0,j}, \hat{s}_{1,j} \in G_q$. Using the public key of main server T_j, the receiver computes each share of x_σ by

$$s_{\sigma,j} = \hat{s}_{\sigma,j} / K_j^\beta.$$

Then the receiver computes

$$x_\sigma = \prod_{j \in \{1,\ldots,t\}} s_{\sigma,j}^{\prod_{l \in \{1,\ldots,t\}}^{l \neq j} \frac{-l}{j-l}}. \tag{6.2}$$

Step 4: The receiver computes, for $b = 0, 1$ and $i = t+1, \ldots, n$,

$$\hat{s'}_{b,i} = \prod_{j \in \{1,\ldots,t\}} \hat{s}_{b,j}^{\prod_{l \in \{1,\ldots,t\}}^{l \neq j} \frac{i-l}{j-l}}.$$

Then he sends $(G_0, G_1, \hat{s'}_{0,i}, \hat{s'}_{1,i})$ to each verification server T_i.

Step 5: Each verification server T_i tests, for $b = 0, 1$,

$$\hat{s'}_{b,i} \quad = \quad s_{b,i} G_b^{k_i}, \tag{6.3}$$

and sends the results of comparisons back to the receiver.

Step 6: If for both $b = 0$ and $b = 1$, more than half of the verification servers reply with "yes" (*i.e.*, reply that (6.3) holds), the receiver accepts the servers' replies and outputs x_σ. Otherwise, the receiver rejects.

6.4 Security Properties of VDOT

Our VDOT protocol has security properties as follows.

Claim 33 *This VDOT protocol is correct.*

Claim 34 *The VDOT protocol protects the receiver's privacy against a coalition of the sender and all the servers.*

Proof: Consider an adversary that controls the sender and all the servers. It is clear that, no matter how adversary cheats, the message sequnce M sent by the receiver follow a distribution that is *symmetric* in σ. That is, $\forall m \in \{0,1\}^*$,

$$Prob[M = m | \sigma = 0] = Prob[M = m | \sigma = 1] = Pr[M = m] = p_m.$$

Suppose that the adversary's output $O_A = O_A(M, \kappa)$, where κ represents the adversary's knowledge. Then,

$$Prob[O_A = \sigma] \quad = \quad Prob[O_A = 0 | \sigma = 0] Prob[\sigma = 0] + Prob[O_A = 1 | \sigma = 1] Prob[\sigma = 1]$$

$$
\begin{aligned}
&= \sum_m Prob[O_A = 0 | \sigma = 0 \wedge M = m] Prob[M = m | \sigma = 0]/2 \\
&\quad + \sum_m Prob[O_A = 1 | \sigma = 1 \wedge M = m] Prob[M = m | \sigma = 1]/2 \\
&= \sum_m \frac{p_m}{2} Prob[O_A(m, \kappa) = 0 | \sigma = 0 \wedge M = m] \\
&\quad + \sum_m \frac{p_m}{2} Prob[O_A(m, \kappa) = 1 | \sigma = 1 \wedge M = m].
\end{aligned}
$$

Because each m is a constant string, and κ is independent of σ,

$$
\begin{aligned}
Prob[O_A = \sigma] &= \sum_m \frac{p_m}{2} Prob[O_A(m, \kappa) = 0 | M = m] + \sum_m \frac{p_m}{2} Prob[O_A(m, \kappa) = 1 | M = m] \\
&= \sum_m \frac{p_m}{2} \\
&= \frac{1}{2}.
\end{aligned}
$$

Claim 35 *Under the DDH assumption, the VDOT protocol protects the sender's privacy against a coalition of the receiver and $t-1$ servers.*

Proof: Note that there is at least one honest main server. Because the receiver's proof has been checked by the honest main server(s), it must be the case that the receiver knows either $\log_g G_0$ or $\log_g G_1$. Suppose that the receiver knows $\log_g G_{\sigma'}$; we will show that the adversary outputs $x_{1-\sigma'}$ with probability less than $\frac{1}{2} + \epsilon$.

First, it is clear that, whenever the adversary computes $x_{1-\sigma'}$, it can derive each share of $x_{1-\sigma'}$. For example, it can derive $s_{1-\sigma',j}$, where T_j is an honest main server. Therefore, it will be sufficient if we can show that the adversary computes $s_{1-\sigma',j}$ with probability less than $\frac{1}{2} + \epsilon$.

To determine the probability that the adversary computes $s_{1-\sigma',j}$, let's look at the adversary's interaction with the honest parties. All the adversary learns from the honest parties is the $\hat{s}_{1-\sigma',j}$s from the honest main servers and the replies indicating

whether $\hat{s'}_{1-\sigma',i} = s_{1-\sigma',i}G_{1-\sigma'}^{k_i}$ from the honest verification servers. The latter can be ignored because the adversary can compute such replies by checking the following identity itself:

$$\hat{s'}_{1-\sigma',i} = \hat{s}_{1-\sigma',j}^{\prod_{l\in\{1,\ldots,t\}}^{l\neq j}\frac{i-l}{j-l}}.$$

However, the former ($\hat{s}_{1-\sigma',j}$s) are ElGamal encryptions of $s_{1-\sigma',j}$s. Because the ElGamal encryption scheme is semantically secure under the DDH assumption, the probability that the adversary computes $s_{1-\sigma',j}$ must be less than $\frac{1}{2}+\epsilon$.

Claim 36 *The VDOT protocol is verifiable if the number of dishonest servers is not more than than* $\frac{n-t}{2}$.

Proof: We construct a verification algorithm V as follows. If the majority of the verification servers reply with "yes," then V outputs "accept;" otherwise, V outputs "reject." It is clear that, if no server cheats, all verification server will reply with "yes" and V will output "accept." In the remainder of this proof, we will show that, when the majority of the verification servers reply with "yes," the receiver's must equal x_σ.

Because the number of dishonest servers is not more than $\frac{n-t}{2}$, among the main servers and the verification servers that reply with "yes," there are at least t that are honest. Suppose that there are h honest main servers and $t-h$ dishonest main servers. Then there are at least $t-h$ honest verification servers that reply with "yes." For each such honest verification server T_i, it must hold that,

$$\hat{s'}_{\sigma,i} = s_{\sigma,i}G_\sigma^{k_i},$$

which implies

$$s_{\sigma,i}G_\sigma^{k_i} = \prod_{j\in\{1,\ldots,t\}} \hat{s}_{\sigma,j}^{\prod_{l\in\{1,\ldots,t\}}^{l\neq j}\frac{i-l}{j-l}}.$$

For $j \in \{1, \ldots, t\}$, we define $\overline{s}_{\sigma,j}$ using $\hat{s}_{\sigma,j} = \overline{s}_{\sigma,j} G_\sigma^{k_j}$. Note that, if T_j is honest, then it must be the case that $\overline{s}_{\sigma,j} = s_{\sigma,j}$. Therefore, we can simplify the above equation as

$$s_{\sigma,i} = \prod_{j \in \{1,\ldots,t\}} \overline{s}_{\sigma,j}^{\prod_{l \in \{1,\ldots,t\}}^{l \neq j} \frac{i-l}{j-l}}.$$

We now have at least $t - h$ $s_{\sigma,i}$s and t $\overline{s}_{\sigma,j}$s that are consistent. Exclude the $t - h$ $\overline{s}_{\sigma,j}$s from dishonest main servers. Then we have at least $t - h$ $s_{\sigma,i}$s and h $\overline{s}_{\sigma,j}$s that are consistent, where each $\overline{s}_{\sigma,j} = s_{\sigma,j}$. These t items uniquely define a degree-$(t-1)$ polynomial, which must be the original polynomial used for secret sharing. The output of the receiver equals this polynomial evaluated at 0, which is exactly x_σ.

6.5 A Protocol for Mobile-Agent Computation

6.5.1 A Global Picture of Mobile-Agent Computation

In this section, we apply VDOT to design a secure protocol for mobile agents. Our system architecture, which is a threshold extension to that in [8], is shown in Figure 6.1.

There are three types of entities in our system architecture: the originator, the hosts, and the servers. The reason for introducing the servers in this mobile computing environment is that the originator may not always be online. Furthermore, because the majority of Internet users are still using dial-up service, they do not have persistent connections. In such scenarios, the servers serve as a proxy to the originator.

Next we briefly discuss each of the entities.

- Originators The responsibility of an originator is to create an agent and send

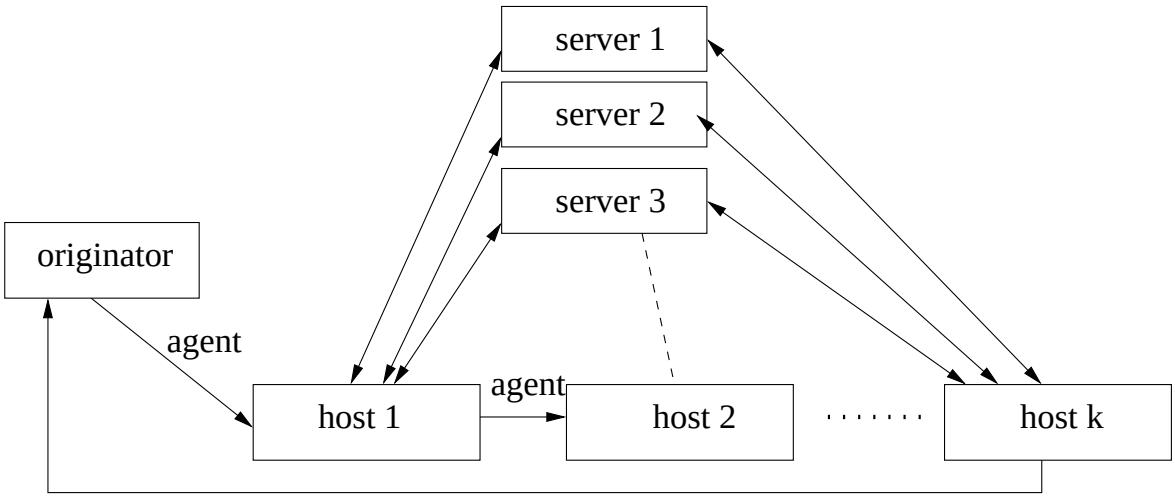

Figure 6.1: System Architecture for Mobile Agent Computation

the agent to the hosts. To improve efficiency, we partition an agent into the security-sensitive portion and the general portion.

- Hosts The responsibility of a host is to run the general portion of an agent and interpret the garbled-circuit portion of the agent. In order to interpret a garbled circuit, the host needs to run the VDOT protocol with the servers to get the appropriate entries from the translation table.

- Servers The responsibility of the servers is to serve as a proxy for an originator and provide translation tables to the hosts through the VDOT protocol.

6.5.2 Protocol Design for Mobile-Agent Computation

The crucial part of our protocol is how to evaluate the security-sensitive function of an agent. Therefore, we start our presentation of the protocol by describing the encoding of the security-sensitive function.

Encoding of a security-sensitive function

For each host, the originator of an agent encodes the security-sensitive function by a garbled circuit and attaches the circuit to the agent. It is proven in [142] that a garbled circuit never reveals any information (to any polynomially bounded adversary) when it is evaluated. However, to evaluate a garbled circuit, a host needs four translation tables:

- (table In1) A table that translates clear input 1 (the previous state) to garbled input 1.
- (table In2) A table that translates clear input 2 (the local input) to garbled input 2.
- (table Out1) A table that translates garbled output 1 to clear output 1 (the new state);
- (table Out2) A table that translates garbled output 2 to clear output 2 (the local output).

Among the four tables, table Out2 is attached to the agent in cleartext so that the host can obtain its local output immediately after the evaluation.

Tables In1 and Out1 encode the state of the agent. Note that the clear output 1 at host j should be the same as the clear input 1 at host $j+1$. A chaining technique [25, 8] is used to combine the entries of table Out1 at host j with the corresponding entries of table In1 at host $j+1$, the next host. Therefore, as long as host j attaches its garbled output 1 to the agent, host $j+1$ is able to obtain its garbled input 1 which corresponds to the agent's state after visiting host j.

Now the only remaining table is In2. For each bit of input 2, the agent originator holds two items —- the garbled inputs for 0 and 1. Note that we must guarantee that

the host receives the item corresponding to its real input bit, but not the other item, because otherwise the host would be able to test the agent with all possible inputs to violate the originator's privacy. So, a VDOT is invoked, with the originator as the sender, the host as the receiver, and the servers as the servers in VDOT. Through this VDOT, the host obtains the garbled input corresponding to its real input bit.

ID	Session identifier
GbCircuit_j	Garbled circuit for host j
GbIn1Host1	Garbled input 1 for host 1
ek_m	The encryption (public) key of the mth server
GbIn1Tab_j(i,b)	The entry of table In1 for host j when the i-th bit of input 1 is b
GbIn2Tab_j(i,b,m)	The m-th share of the entry of table In2 for host j when the i-th bit of input 2 is b
GbOut1Tab_j(i,b)	The entry of table Out1 for host j when the i-th bit of output 1 is b
GbOut2Tab_j	The translation table Out2 for host j

Table 6.1: Notations in Figure 6.2

Figure 6.2 summarizes the data format carried by an agent for a security-sensitive function (the notation is explained in Table 6.1). In our protocol, we use both asymmetric encryption and symmetric encryption. Here, we denote by $PE(ek, m)$ the asymmetric encryption of cleartext m with encryption key ek; we denote by $E(k, m)$ the symmetric encryption of cleartext m with key k. In our protocol, we require that it be easy to verify whether or not a ciphertext is encrypted with a key in the symmetric encryption scheme. Note that this property can be implemented by adding redundancy to the cleartext before encryption.

Figure 6.2: Data Format of a Security-Sensitive Function in an Agent

Protocol Summary for Mobile-Agent Computation

When an agent arrives at a host, since In1 is chained to Out1 of the previous host, the host uses the garbled output 1 of the previous host to retrieve its garbled input 1. The host then executes VDOT to obtain the value of garbled input 2 corresponding to its local input.

With both garbled input 1 and garbled input 2, the host evaluates the garbled circuit. After the evaluation, the host uses the attached table Out2 to get its local output. Then it attaches its garbled output 1 to the agent so that the next host can retrieve its garbled input 1 from the agent.

Figure 6.3 shows the information flow of our protocol at a host.

The last host sends the agent back to the originator. The originator then translates the garbled output 1 to determine the final state of the computation.

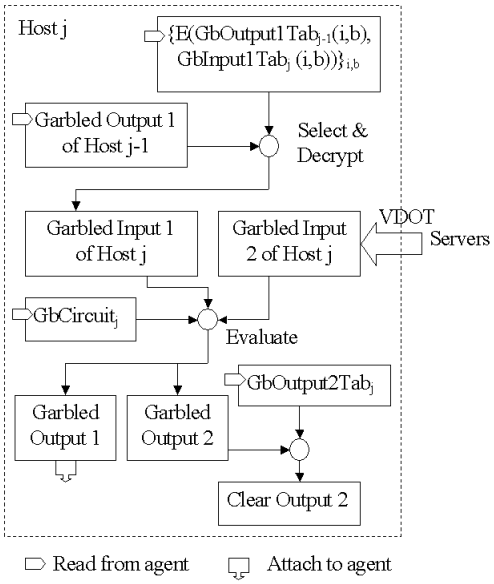

Figure 6.3: Evaluating a Security-Sensitive Function at Host j

6.5.3 Security Analysis of the Mobile-Agent Protocol

We briefly analyze the security properties of our protocol for mobile-agent computation.

- Originator's Privacy The originator's private information in the security-sensitive portion of the agent is private against any hosts and any servers, unless t or more servers collude.

  This follows from the property of sender's privacy of VDOT. Because only the originator knows the translation tables of input 1 and output 1, the state information (in which the originator's private information is hidden) is private against other parties. Because VDOT ensures that each host can only evaluate the garbled circuit with one value of its local input, no host is able to extract partial private information from the garbled circuit by evaluating it for multiple times.

- Host's Privacy A host's local input to the agent and local output from the

agent are private against the originator, any other hosts and any servers, no matter how many parties involved collude.

The privacy of local input follows from the property of receiver's privacy of VDOT. Because the local input is not revealed in VDOT, there is no way for other parties to learn about it. The privacy of the local output is obvious.

- Cheating Detection If the servers cheat to change the garbled input 2, the host is able to detect cheating, unless $\frac{n-t}{2}$ or more servers collude.

  This follows from the verifiability of VDOT.

6.6 Implementation and Performance Evaluation

We have implemented VDOT and garbled circuits, the key components of system. We have also measured preliminary performance.

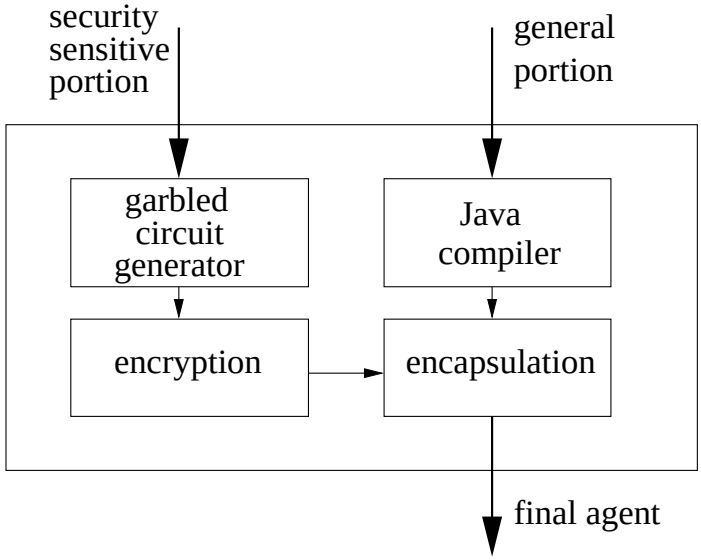

Figure 6.4: Software Architecture of an Originator

In our software design, the general portion of an agent will be implemented in Java, while the security-sensitive portion will be encoded as a garbled circuit. Fig-

ure 6.4 shows the components and the information flow at an originator. In our current implementation, a user needs to manually generate a garbled circuit, which should be very small for many applications. In the future, we expect that an automatic circuit generator will be built. Using the automatic circuit generator, a user can generate a circuit for her own use by specifying her own parameters. For example, to build an agent that searches for airline tickets, all a user needs to do is to execute the generator and input her desired flight date, source, destination and price threshold. Then the generator immediately outputs a mobile agent on her behalf.

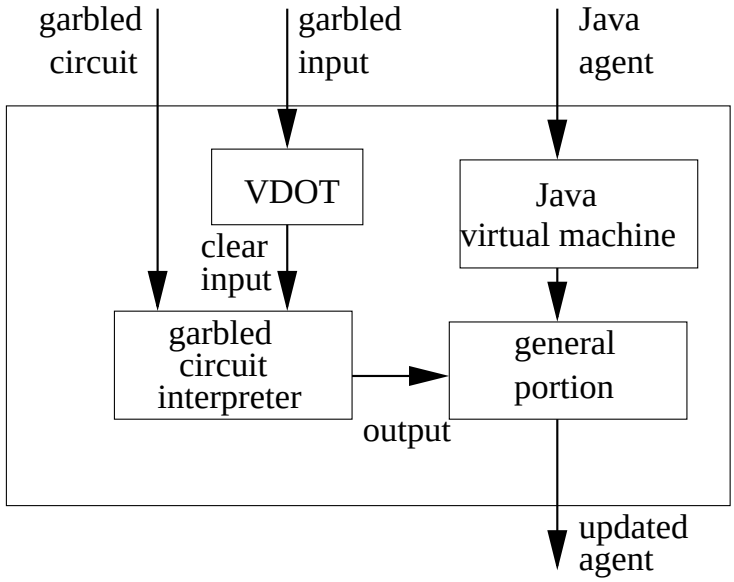

Figure 6.5: Components of a Host

An agent is sent to hosts for execution. Figure 6.5 shows the components and the information flow at a host. Because garbled circuits are general purpose and are represented in a platform-independent format, for the purpose of efficiency, our current interpreter is implemented in C.

Obviously, one potential major overhead will be the evaluation of garbled circuits. However, measurement of our prototype interpreter shows that the overhead is very small. Figure 6.6 shows the overhead of evaluating random garbled circuits

of different sizes. The result shows that the overhead of evaluating a garbled circuit of several hundred gates is pretty small.

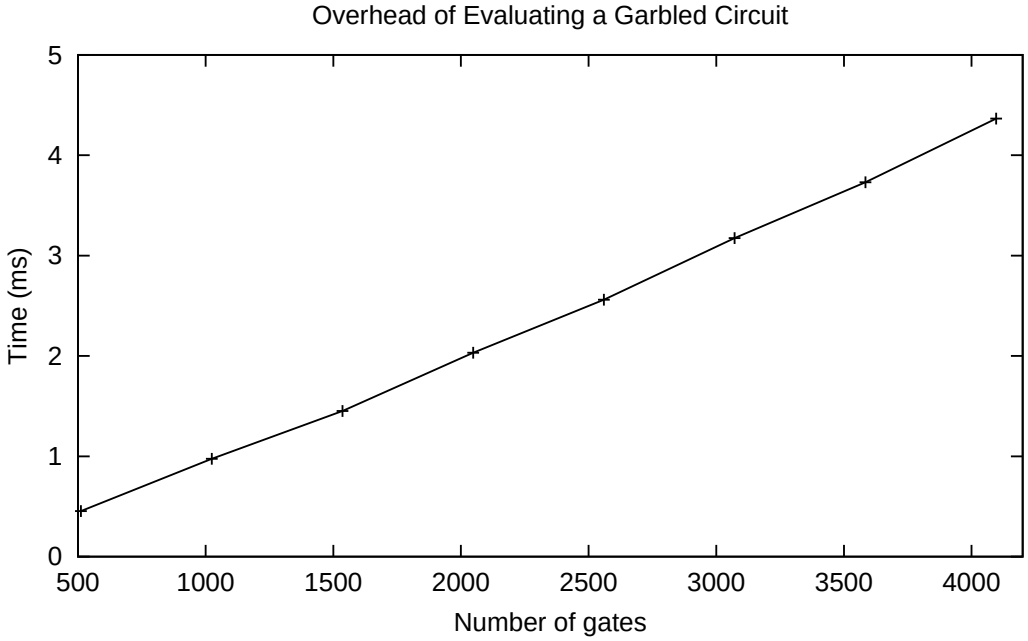

Figure 6.6: Overhead of Evaluating a Garbled Circuit

In order to interpret garbled circuits, the hosts need to interact with the servers through the VDOT protocol to retrieve garbled input 2. Our prototype of VDOT is implemented in C++. We evaluate the overhead of the VDOT protocol on machines with Intel 1.0GHz CPU running Linux. Figure 6.7 shows the steps of the VDOT protocol and labels the cost of each computational step. The setting of the evaluation is $n = 6$ and $t = 3$. It is clear that the cost of VDOT is acceptable.

6.7 Summary of Secure Mobile-Agent Computation

In this chapter, we propose a cryptographic primitive VDOT and present a concrete design of VDOT. We showed that our design is *correct* and it protects both the

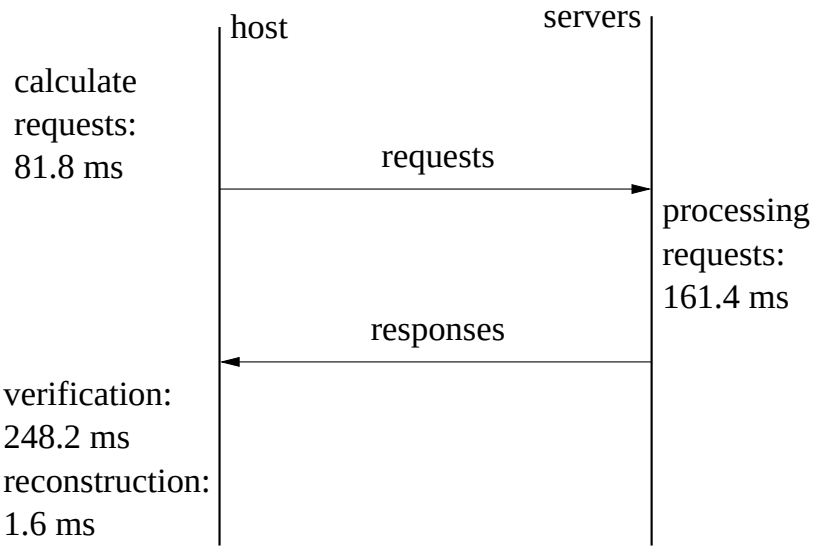

Figure 6.7: Overhead of VDOT ($(n, t) = (6, 3)$)

receiver's privacy and the *sender's privacy*. It also has *verifiability* such that cheating can be detected if servers cheat to change the receiver's output. This work can be viewed as an application of the secure multi-party computation techniques shown in Chapter 2. However, we also use other techniques like consistency verification of encrypted shares.

We apply VDOT to design a protocol for secure mobile-agent computation. Our system partitions an agent into the general portion and the security-sensitive portion. Our system protects the privacy of both the originator and the hosts, without using any single trusted party. We have also implemented the key components of our system. As far as we know, this is the first effort to implement a system that protects the privacy of mobile agents. Our preliminary evaluation shows that protecting mobile agents is not only possible, but also can be implemented efficiently.

Chapter 7

Incentives in Ad Hoc Networks

In previous chapters, I have considered problems of privacy and integrity in various scenarios. In this chapter, I present joint work with Jiang Chen and Yang Richard Yang [144], which studies the problem of incentives in mobile ad hoc networks. This is an attempt to add incentive considerations to secure multi-party computation. Specifically, in this work, we study how to stimulate cooperation among selfish mobile nodes in an ad hoc network. We propose Sprite, a simple, cheat-proof, credit-based system for stimulating cooperation among selfish nodes in mobile ad hoc networks. Our system suppresses cheating behavior and provides incentive for mobile nodes to cooperate and report actions honestly. Compared with previously proposed solutions to this problem, our system does not require any tamper-proof hardware at any node. Furthermore, we present a formal model of our system and prove its properties in this model. Evaluations of a prototype implementation show that the overhead of our system is small. Simulations and analysis show that mobile nodes can cooperate and forward each other's messages, unless the resource of each node is extremely low.

7.1 Background and Motivation

In recent years, mobile ad hoc networks have received much attention due to their potential applications and the proliferation of mobile devices [115, 137]. Specifically, mobile ad hoc networks refer to wireless multi-hop networks formed by a set of mobile nodes without relying on a preexisting infrastructure. In order to make an ad hoc network functional, the nodes are assumed to follow a self-organizing protocol, and the intermediate nodes are expected to relay messages between two distant nodes. Recent evaluations have shown that ad hoc networks not only are flexible and robust, but also can have good performance in terms of throughput, delay and power efficiency [72].

So far, applications of mobile ad hoc networks have been envisioned mainly for emergency and military situations. In such applications, all the nodes of the network belong to a single authority and therefore have a common objective. As a result, cooperation among the nodes can be assumed. However, as observed by several authors [93, 23, 24, 21, 22], it may soon be possible to deploy ad hoc networks for civilian applications as well. In such emerging civilian applications, the nodes typically do not belong to a single authority. Consequently, cooperation behaviors such as forwarding each other's messages, cannot be assumed directly.

We distinguish two types of uncooperative nodes: faulty/malicious nodes and selfish nodes. Although the problems of faulty/malicious nodes can be important in military applications, the focus of this work is on selfish nodes, which we expect will be the dominant type of nodes in a civilian ad hoc network.[1] Specifically, a selfish node is an economically rational node whose objective is to maximize its own welfare,

[1]The problems of faulty/malicious nodes can be addressed from many layers, for example, using spread-spectrum encoding to avoid interference over the communication channel; using a reputation system to identify the faulty/malicious nodes and subsequently avoid or penalize such nodes [93]; and applying the techniques of secure multi-party computation to perform computation correctly even in the presence of faulty/malicious nodes.

which is defined as the benefit of its actions minus the cost of its actions. Because forwarding a message will incur a cost (of energy and other resources) to a node, a selfish node will need incentive in order to forward others' messages.

One possibility to provide incentive is to use a reputation system [93, 21, 22]. For example, in [93], Marti et al. proposed a reputation system for ad hoc networks. In their system, a node monitors the transmission of a neighbor to make sure that the neighbor forwards others' traffic. If the neighbor does not forward others' traffic, it is considered as uncooperative, and this uncooperative reputation is propagated throughout the network. In essence, we can consider such a reputation system as a repeated game whose objective is to stimulate cooperation (*e.g.*, see Chapter 8 of [110]). Such reputation systems, however, may have several issues. First, to the best of our knowledge, there is no formal specification and analysis of the type of incentive provided by such systems. Second, these systems have not considered the possibility that even selfish nodes can collude with each other in order to maximize their welfare. Third, some of the current systems depend on the broadcast nature of wireless networks in order to monitor other nodes. Such monitoring, however, may not always be possible due to asymmetric links when nodes use power control. Furthermore, directional antennas [131, 140], which are gaining momentum in wireless networks in order to improve capacity, will also make monitoring hard.

Another possibility to provide incentive is to use credit (or virtual currency). Buttyan and Hubaux proposed a nice solution of this type in [23], and then presented an improved result based on credit counters in [24]. For both proposals, a node receives one unit of credit for forwarding a message of another node, and such credits are deducted from the sender (or the destination). Besides other potential issues that we will discuss in Section 7.2, both proposals require a piece of tamper-proof hardware at each node so that the correct amount of credit is added or deducted from

the node. As a result of this requirement, although both proposals are interesting, they may not find wide-spread acceptance.

In this work, we propose Sprite, a simple, cheat-proof, credit-based system for mobile ad-hoc networks with selfish nodes. Similar to [23] and [24], our system also uses credit to provide incentive to selfish nodes. However, one of the novel and distinguishing features is that our system does not need any tamper-proof hardware at any node.

At a high level, the basic scheme of our system can be described as follows. When a node receives a message, the node keeps a *receipt* of the message. Later, when the node has a fast connection to our Credit Clearance Service (CCS), it reports to the CCS the messages that it has received/forwarded by uploading its receipts. The CCS then determines the charge and credit to each node involved in the transmission of a message, depending on all the reported receipts of a message.

There are two issues regarding the design of our system. First, because there is no tamper-proof hardware at any node and the charge and credit are based on the reports of the selfish nodes, a selfish node (or even a group of colluding node) may attempt to cheat the system to maximize its expected welfare. As an example, a selfish node may withhold its receipt, or collude with other nodes to fake receipts, if such actions can maximize its welfare. This is the security perspective of our system. Second, a node should receive enough credit for forwarding a message for another node, so that it can send its own messages with the received credit, unless the resource of the node itself is extremely low. This is the incentive perspective of our system.

In summary, the contributions of this work are the following. First, we present Sprite, a system to provide incentive to selfish mobile nodes to cooperate. Second, our system determines charge and credit from a game-theoretic perspective, and

motivates each node to report its actions honestly, even when a collection of the selfish nodes collude. Third, we model the essential component of our system as a game and prove the correctness of our system under this model. As far as we know, this is the first pure-software solution that has formal proofs of security. Our main result works for message-forwarding, and we extend it to route discovery as well. Fourth, we perform extensive evaluations and simulations of our system. Evaluations of a prototype implementation show that the overhead of our system is small. Simulations show that the nodes will cooperate and forward each other's messages, unless the resource of each node is extremely low.

The rest of this chapter is organized as follows. In Section 7.2, we discuss works related to this problem. In Section 7.3, we present the overall architecture and the intuitions behind our design. We then give the full specification our system in Section 7.4. In Section 7.5, we present a formal model of our system and prove the security properties under this model. In Section 7.6, we extend our work to route discovery. In Section 7.7, we present evaluations of our solution. We summarize this work in Section 7.8.

7.2 Related Work

There are three classes of work closely related to ours: reputation systems, stimulation approaches from the Terminodes project, and game-theoretic research in computer science.

7.2.1 Reputation-based Approaches

In [93], Marti et al. considered uncooperative nodes in general, including selfish and malicious nodes. In order to cope with this problem, they proposed two tools: a

watchdog, which identifies misbehaving nodes, and a pathrater, which selects routes that avoid the identified nodes. Their simulations showed that these two tools can maintain the total throughput of an ad hoc network at an acceptable level even with a large percentage of misbehaving nodes. In [21, 22], Buchegger and Le Boudec proposed and evaluated their CONFIDENT protocol, which detects and isolates misbehaving nodes. However, as we discussed in Section 7.1, there are several issues that such reputation-based systems need to address.

In [84], Jakobsson, Wetzel, and Yener show that any routing algorithm in ad hoc networks can be attacked with a certain cost. They also suggest to use *reputation-base control* to immunize against these attacks.

7.2.2 Stimulation Approaches from Terminodes

In this subsection, we review several pieces of related work where all or some of the authors are from the Terminodes project. General reviews of the Terminodes project, and of the related security problems, can be found in [74, 76, 75].

In [23], Buttyan and Hubaux proposed a stimulation approach that is based on a virtual currency, called nuglets, which are used as payments for packet forwarding. Using nuglets, the authors proposed two payment models: the Packet Purse Model and the Packet Trade Model. In the Packet Purse Model, the sender of a packet pays by loading some nuglets in the packet before sending it. Intermediate nodes acquire some nuglets from the packet when they forward it. If the packet runs out of nuglets, then it is dropped. In the Packet Trade Model, the destination of a packet pays for the packet. To implement the Packet Trade Model, each intermediate node buys a packet from its previous node for some nuglets and sells it to the next node for more nuglets. In this way each intermediate node earns some nuglets and the total cost of forwarding the packet is covered by the destination. To implement either the Packet

Purse Model or the Packet Trade Model, a tamper-proof hardware is required at each node to ensure that the correct amount of nuglets is deducted or credited at each node.

Besides the requirement for a tamper-proof hardware at each node, some other issues also exist for the Packet Purse Model and the Packet Trade Model:

1. Both models require the clearance of nuglets in real-time. As a result, if the system does not have enough nuglets circulating around, the performance of their system may degrade.

2. Under both models, if a mobile node runs out of nuglets, its tamper-proof hardware still has to contact with some central authority in order to "refill" its credit. (Actually, the CSS introduced by our system is similar to such an authority.)

3. A disadvantage of the Packet Trade Model is that it is vulnerable to network overload, because the senders do not have to pay. For this reason, the authors of [23] mainly studied the Packet Purse Model.

Besides the nuglet approach, Buttyan and Hubaux also proposed a scheme based on credit counter [24]. In this new approach, each node keeps track of its remaining battery and its remaining credit. The authors simulated four rules for a node to determine when to forward others' packets and when to send its own packets. Our analysis shows that the first rule is actually optimal to achieve their given goals. Although this new scheme is simple and elegant, it still requires a tamper-proof hardware at each node so that the correct amount of credit is deducted or credited. Furthermore, the first two issues we outlined in the previous paragraph exist for this approach as well.

In [80], Jakobsson, Hubaux, and Buttyan study the incentive issue in both routing and packet forwarding in multi-hop cellular networks. Their solution is an integrated approach that *avoids separation of routing and forwarding* and encourages nodes' cooperation using micropayment. Their design not only detects and rewards collaboration, but also detects and punishes various forms of abuse. Because they use light-weight cryptographic techniques, their scheme is highly efficient. Compared to our work, [80] is in a related-but-different scenario (hybrid cellular networks that include base stations in addition to ad hoc networks, with multi-hop uplinks and one-hop downlinks), and using a different approach (micropayment, not transaction clearance at a CCS).

In [17], Ben Salem, Buttyan, Hubaux, and Jakobsson propose another very efficient scheme for packet forwarding in multi-hop cellular networks (where both uplinks and downlinks are one-hop). This solution is session-based and independent of routing. Thus, it can be used in combination with existing secure routing schemes.

7.2.3 Related Work in Algorithmic Mechanism Design and Game Theory

Our approach is motivated by algorithmic mechanism design (see *e.g.*, [106, 105, 112, 68, 45, 65, 44, 123]), which is an emerging active research area in the intersection of computer science and mathematical economics. In particular, Feigenbaum et al. have considered BGP-based mechanism design for lowest-cost unicast routing in the Internet [44]. In [45], Feigenbaum et al. have considered cost sharing for multicast. Golle et al. have analyzed the incentives in peer-to-peer networks [65]. However, as far as we know, before [144] there was no previous proposed mechanism design for ad hoc networks. Furthermore, although our design is motivated by algorith-

mic mechanism design, our problem does not fit exactly into the mechanism-design framework. For example, in our game, the information held by each player is not totally private, while in mechanism design, each player must have a private *type*. Furthermore, algorithmic mechanism design only considers economic incentives and does not address how to enforce the output of a mechanism. In contrast, our solution enforces a payment scheme using a cryptographic technique.

Qiu, et al. [118] and Srinivasan, et al. [132] also study incentives in wireless ad hoc networks. They focus on the economic perspective of the problem using game-theoretic approaches and ignore how to enforce the mechanism they design.

7.3 Overview of our Approach

In this section, we present the overall architecture and the intuitions behind our design; the formal results will be presented in Sections 7.4 and 7.5.

7.3.1 System Architecture

Figure 7.1 shows the overall architecture of our system, which consists of the Credit Clearance Service (CCS) and a collection of mobile nodes. The nodes are equipped with network interfaces that allow them to send and receive messages through a *wireless overlay* network [133], *e.g.*, using GPRS in a wide-area environment, while switching to 802.11 or Bluetooth in an environment where high-bandwidth Internet access is available. To identify each node, we assume that each node has a certificate issued by a scalable certificate authority such as those proposed in [146, 91]. For concreteness of presentation, we assume that the sender knows the full path from the sender to the destination, using a secure ad hoc routing protocol based on DSR [85, 37, 73]. The incentive issues of route discovery will be investigated in Section 7.6.

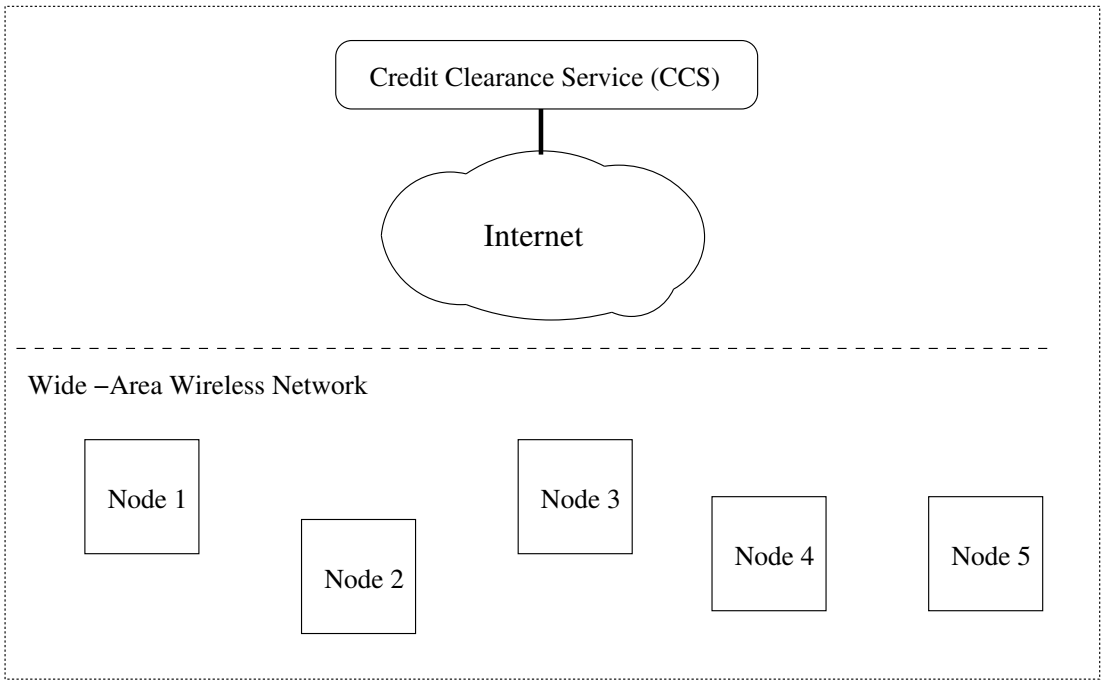

Figure 7.1: The architecture of Sprite.

When a node sends its own messages, the node (or the destination, see later) will lose *credit* (or virtual money) to the network because other nodes incur a cost to forward the messages. On the other hand, when a node forwards others' messages, it should gain credit and therefore be able to send its messages later.

There are two ways for a node to get more credit. First, a node can pay its debit or buy more credit with real money, at a variable rate to the virtual money, based on the current performance of the system. However, the preferred and dominant way to get more credit is by forwarding others' messages. In order to get credit for forwarding others' messages, a node needs to report to the CCS which messages it has helped to forward. Although a node can save its reports in a local storage such as CompactFlash card, in order to reduce storage, each mobile node should report to the CCS whenever it switches to a fast connection and has backup power. A mobile node can also use a desktop computer as a proxy to report to the CCS. In order to save bandwidth and storage, instead of requiring the whole message as a report,

our system uses small *receipts*. Such receipts are derived from the content of the messages but do not expose the exact content of the messages. Thus, although we require that the CSS be trusted in terms of maintaining credit balance, the nodes do not need to trust the CSS in terms of message confidentiality.

7.3.2 Who Pays Whom?

Before determining the amount of credit or charge to each node, we first discuss two basic questions.

The first question is who pays whom. Considering the relay of a message from a sender to a destination as a transaction, we need to decide who should be charged for the message and who should receive credit for relaying the message.

Although we can charge the destination, we decide that charging the sender will be a more robust and general approach. There are two reasons for charging the sender only. First, charging the destination may allow other nodes to launch a denial-of-service attack on the destination by sending it a large amount of traffic. Even sharing the cost between the sender and the destination could have a similar problem, because the sender could collude with the intermediate nodes, who could secretly return the sender's payment back, so that only the destination pays for the traffic. On the other hand, if only the sender is charged, a node will not have incentive to send useless messages. Second, if the destination benefits from the content of a message and thus should pay for it, the sender can get compensation from the destination, for example, through an application-layer payment protocol. Given these reasons, only the sender will be charged in our system.

A closely related question is who will receive credit for forwarding a message. Ideally, any node who has ever tried to forward a message should be compensated because forwarding a message will incur a cost to the node, no matter successful

or not. However, a forwarded message may be corrupted on the link, and there is no way to verify that the forwarding action does occur. Although some local wireless networks such as IEEE 802.11 do provide link layer acknowledgments, such acknowledgment schemes are not universal and we refrain from changing basic network functions. Given this decision, the credit that a node receives will depend on whether or not its forwarding action is successful — a forwarding is successful if and only if the next node on the path receives the message. In other words, the CCS believes that a node has forwarded a message if and only if there is a successor of that node on the path reporting a valid receipt of the message.

7.3.3 Objectives of the Payment Scheme

The second basic question is about the objective of the payment scheme. After all, the objectives of our payment scheme are to prevent cheating actions and to provide incentive for the nodes to cooperate. Given such objectives, our system does not target *balanced payment*; that is, we do not require that the total charge to the sender be equal to the total credit received by other nodes for a message. In stead, we require that the system be *budget sufficient*, which means that the overall charge equals the total credit if nobody cheats, and that the overall charges is greater than or equal to the total credit if some party cheats. In fact, in order to prevent one type of cheating actions, when that type of cheating happens, our CCS charges the sender more than it gives to the other nodes (see Subsection 7.3.6). In order to offset long-term net outflow of credit from the mobile nodes to the CCS, if in a large network, the CCS periodically returns the credit back to the mobile nodes uniformly; otherwise, the CCS periodically gives each mobile node a fixed amount of credit. Note that this return will not enable any cheating action or reduce the incentive of the nodes to forward others' messages.

7.3.4 Cheating Actions in the Receipt-Submission Game

Because the mobile nodes are selfish, without a proper payment scheme, they may not forward others' messages or they may try to cheat the system, if the cheating can maximize their welfare. In particular, a selfish node can exhibit one of the three selfish actions:

1. After receiving a message, the node saves a receipt but does not forward the message;

2. The node has received a message but does not report the receipt;

3. The node does not receive a message but falsely claims that it has received the message.

Note that any of the selfish actions above can be further complicated by collusion of two or more nodes. We next progressively determine the requirements on our system in order to prevent the above actions.

7.3.5 Motivating Nodes to Forward Messages

In order to motivate a selfish node to forward others' messages, the CCS should give more credit to a node who forwards a message than to a node who does not forward a message. A basic scheme to achieve this objective is as follows. First, the CCS determines the last node on the path that has ever received the message. Then the CCS asks the sender to pay β to this node, and α to each of its predecessors, where $\beta < \alpha$. Note that the CCS does not ask the sender to pay anything to the successors of the last node. Comparing this scheme with those in [23] and [24], we observe that the approaches in [23] and [24] are just the special case that β is very small and α is close to 1. Figure 7.2 illustrates the basic idea with an example. In this example,

only the first three intermediate nodes submit their receipts. Therefore, nodes 1 and 2 will each receive a payment of α, and node 3 a payment of β. Because node 4 and the destination do not submit any receipt, they do not receive any credit. The sender pays a total of $2\alpha + \beta$.

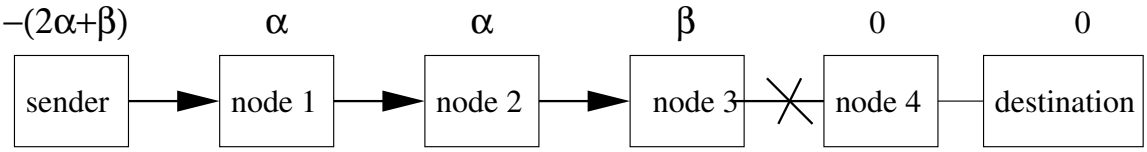

Figure 7.2: Illustration of our payment scheme (version 1).

7.3.6 Motivating Nodes to Report their Receipts

Obviously, each single node having received a message is motivated to report its receipt, if β is greater than its cost of submitting a receipt, which, as we discussed previously, should be low because a receipt is generally small.

Unfortunately, there is still a collusion that can work against the above design. As an example, the last node (or in the general case, the last k nodes) ever received the message can collude with the sender. In particular, if the last node does not report its receipt, the sender saves α while the last node loses β. However, if the sender gives the last node a behind-the-scene compensation of $\beta + \epsilon$, where $\epsilon > 0$, the last node will be better-off while the sender still enjoys a net gain of $\alpha - (\beta + \epsilon)$. Thus, the colluding group gets a net benefit of $\alpha - \beta$.

In order to prevent this cheating action, the CCS charges the sender an extra amount of credit if the destination does not report the receipt for message. This extra charge goes to the CCS instead of any nodes. The overall charge to the sender (including payments to other nodes and the extra charge) should be $k\beta$ less than the charge to the sender when the message arrives at the destination, where k is the

number of nodes not submitting receipts. Given such extra charge, even a colluding group cannot benefit from this cheating action. Again consider the example in Figure 7.2. Figure 7.3 shows the revised amount paid by the sender, which is equal to $(4\alpha + \beta) - 2\beta)$.

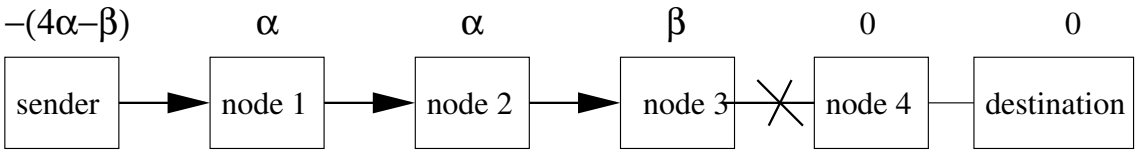

Figure 7.3: Illustration of our payment scheme (version 2).

7.3.7 Preventing False Receipts

Next we consider a countermeasure to the third type of selfish actions. As we discussed before, in order to save bandwidth and storage, our system requires that the nodes submit receipts instead of full messages. Given such a scheme for receipts, a group of colluding nodes can try to attack our system in several ways. For example, instead of forwarding the whole message, an intermediate node can forward only the receipt of a message to its successor, which is sufficient for getting credit. Moreover, the intermediate node can even wait until it has a fast connection to the successor to forward the false receipt, thus further saving resource usage.

The key to prevent such attack depends on the destination. We distinguish two cases: 1) the destination colludes with the intermediate nodes; or 2) the destination does not collude with the intermediate nodes.

We first consider the case that the destination colludes with the intermediate nodes, and therefore submits a receipt of a message even when it does not receive the whole message. For this case, we argue that the intermediate nodes and the destination should be paid as if no cheating had happened, because after all, the

message is for the destination and the destination does submit a receipt for the message, indicating that it has received the message. If the sender needs to make sure that the destination receives the whole message, a higher-layer protocol to validate the receipt of the whole message by the destination can be easily implemented (see, *e.g.*, [127]).

We next consider the case that the destination does not collude with the intermediate nodes. In this case, if the intermediate nodes forward only the receipt of a message instead of the whole message, then the destination will not be able to receive a valid message payload, and therefore will not submit a receipt for the message. Based on this observation, we can prevent the potential cheating action of the intermediate nodes by greatly reducing the amount of credit given to the intermediate nodes, if the message is not reported by the destination. With such reduction of credit, the cheating nodes cannot get enough credit even to cover the minimum expense needed for this type of cheating, *i.e.*, the cost of forwarding a receipt. To be more exact, if the destination does not report a receipt of a message, we multiply the credit paid to each node by γ, where $\gamma < 1$ (the exact requirement on γ will be presented in Section 7.5). Still consider the example in Figure 7.2. Figure 7.4 shows the revised amount of credit received by each node. In particular, comparing Figure 7.4 with Figure 7.3, due to this revision, we reduce the charge to the sender by $\gamma\beta$ instead of β, for each node on the path who does not report a receipt.

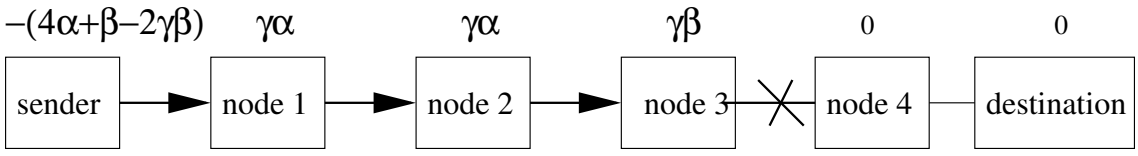

Figure 7.4: Illustration of our payment scheme (final version).

7.4 Specification of the Message-Forwarding Protocol

In the following formal specification of our protocol, we denote the public/private key pair of node n_i by (PK_i, SK_i). Each node n_i maintains a sequence-number matrix seq_i, where $seq_i(j,k)$ is the sequence number of messages from sender n_j to destination n_k, observed by node n_i. We assume that $(sign_{SK}(), verify_{PK}())$ is a digital signature scheme. In practice, we can use the RSA or the elliptic curve signature scheme.

7.4.1 Sending a Message

Suppose that node n_0 is to send message payload m with sequence number $seq_0(0,d)$ to destination n_d, through path p. Node n_0 first computes a signature, s, on $(MD(m), p, seq_0(0,d))$, where $MD()$ is a message digest function such as MD5 [120] or SHA-1 [107]. Then, n_0 transfers $(m, p, seq_0(0,d), s)$ to the next hop and increases $seq_0(0,d)$ by 1. Figure 7.5 specifies the complete protocol steps.

$\triangleright$ m is the message payload.
$\triangleright$ n_0 is the sender, n_d the destination, and p the path.

$s \leftarrow sign_{SK_0}(MD(m), p, seq_0(0,d))$
send $(m, p, seq_0(0,d), s)$ to the next node
$seq_0(0,d)++$

Figure 7.5: Node n_0 sends a message to n_d.

7.4.2 Receiving a Message

Suppose that node n_i receives (m, p, seq, s). It first checks three conditions: 1) n_i is on the path; 2) the message has a sequence number greater than $seq_i(0,d)$; and 3) the

signature is valid. If any of the conditions is not satisfied, the message is dropped. Otherwise, it saves $(MD(m), p, seq, s)$ as a receipt. If n_i is not the destination and decides to forward the message, it sends (m, p, seq, s) to the next hop. Figure 7.6 specifies the protocol steps.

```
▷ (m, p, seq, s) is the received message.
▷ n_0 is the sender, n_d the destination.

if ((n_i not in p) || (seq ≤ seq_i(0, d))
    || (verify_{PK_0}((MD(m), p, seq), s) ≠ TRUE))
   drop the message
else
   seq_i(0, d) ← seq
   save (MD(m), p, seq, s) as a receipt
   if (n_i is not the destination and decides to forward)
      send (m, p, seq, s) to next hop
   else
      drop the message
```

Figure 7.6: Node n_i receives (m, p, seq, s).

7.4.3 Computing Payments

A receipt (D, p, seq, s) submitted by node n_i is regarded as valid if

$$verify_{PK_0}((D, p, seq), s) = TRUE,$$

where PK_0 is the public key of the sender.

Without loss of generality, we assume that $p = (n_0, n_1, \ldots, n_e, \ldots, n_d)$, where n_e is the last node on path p that submits a valid receipt with sequence number seq. Then the CCS charges C from node n_0, and pays P_i to node n_i, where

$$C = (d-1)\alpha + \beta - (d-e)\gamma\beta,$$

$$P_i = \begin{cases} \alpha & \text{if } i < e = d \\ \beta & \text{if } i = e = d \\ \gamma\alpha & \text{if } i < e < d \\ \gamma\beta & \text{if } i = e < d. \end{cases}$$

Note that in implementation, the CCS will issue credit gradually. For example, when the last intermediate node submits its receipt for a message but the destination has not submitted its receipt yet, the last intermediate node will get $\gamma\beta$. Later, when the destination submits its receipt, the node will get its full credit of α.

7.5 Formal Model and Analysis of the Message-Forwarding Protocol

7.5.1 A Model of the Receipt-Submission Game

For convenience of analysis, we model the submissions of receipts regarding a given message m as a one-round game. Note that the whole system can also be viewed as a game, which is much more complicated. However, here we focus on the component of receipt submissions, so that we can have a thorough analysis.

Players. This game has $d+1$ players, $n_0, n_1, \ldots, n_d$, from the sender to the destination.[2]

Players' Information. Let T_i be the information held by player n_i that is unknown to the CCS. For $i > 0$, $T_i = TRUE$ if node n_i has ever received message m; $T_i = FALSE$ otherwise. Obviously, the sender n_0 and the set of nodes that have

[2]Recall that each receipt contains the signed path. Therefore, nodes not on the path are easily excluded from this game.

ever received message m constitute a prefix of the path. Therefore,

$$T_i = \begin{cases} TRUE & \text{if } 0 < i \leq e' \\ FALSE & \text{if } e' < i \leq d, \end{cases}$$

where e' is the index of the last node that has ever received message m. Note that e' is secret to the CCS when the game starts. Also note that a player has some partial information about e', *i.e.*, the information inferred from its own information. For completeness, we define $T_0 = TRUE$.

Actions. Each player, n_i ($i > 0$), has two possible actions: reporting that it has ever received message m (by submitting a valid receipt), or withholding its report. We denote the action of n_i by A_i. Then A_i is either $TRUE$ or $FALSE$. The only exception is n_0, which has no choice of action. We define $A_0 = TRUE$, for completeness of our model.

Cost of Actions. We denote the cost of n_i's action by U_i. As discussed before, in general, the cost of sending a receipt to the CCS is very low. However, if player n_i does not receive message m but can successfully claim that it has received the message, then a colluding node must have forwarded n_i a copy of the receipt. Let δ be the cost of forwarding a receipt from one mobile node to another node. Then the colluding node incurs a cost of δ and n_i must compensate the colluding node with δ. Counting this cost on n_i, we have

$$U_i = \begin{cases} \delta & \text{if } T_i = FALSE \text{ and } A_i = TRUE \\ 0 & \text{otherwise.} \end{cases}$$

Payment. Recall that the system's payment to n_i ($i > 0$) is

$$P_i = \begin{cases} \alpha & \text{if } i < e = d \\ \beta & \text{if } i = e = d \\ \gamma\alpha & \text{if } i < e < d \\ \gamma\beta & \text{if } i = e < d. \end{cases}$$

For n_0, the charge of C can be viewed as a negative payment

$$P_0 = -C = -((d-1)\alpha + \beta - (d-e)\gamma\beta).$$

Welfare. For player n_i, deducting its cost from its received payment, the node has a welfare of

$$W_i = P_i - U_i.$$

7.5.2 Security Analysis of the Receipt-Submission Game

If $A_i = T_i$, then we say that n_i *tells the truth.* Otherwise, we say that n_i *cheats.* The *strategy* of n_i can be *truth-telling*, *cheating*, or a probability distribution over these two choices. The *strategy profile* of a group of players refers to the ordered set of the strategies of these players.

Definition 37 *For a player, an* optimal strategy[3] *is a strategy that brings the maximum expected welfare to it,* regardless of the strategies of all the other nodes.

Theorem 38 *In the receipt-submission game, truth-telling is an optimal strategy for every node n_i, if $\delta \geq \gamma\beta$, and n_d does not cheat in case of $T_d = FALSE$.*

[3]Note that an *optimal strategy* is not necessary a *dominant strategy.* See, *e.g.*, [110] for the definition of dominant strategy.

Proof: Consider a strategy profile of all of the rest players, in which each player, n_j ($j \neq i$), tells the truth with probability x_j. We distinguish four cases here.

- Case A. $i = 0$. Because $A_i = T_i = TRUE$ is the only possible strategy, it is also the best response.

- Case B. $0 < i < e'$. Recall that e' is the index of the last node that has ever received the message. If n_i tells the truth, its expected welfare $EW_i^+ = EP_i^+$ is

$$\begin{cases} (1 - \prod_{j=i+1}^{e'}(1 - x_j) \prod_{j=e'+1}^{d-1} x_j)\gamma\alpha \\ + \prod_{j=i+1}^{e'}(1 - x_j) \prod_{j=e'+1}^{d-1} x_j\gamma\beta \\ \qquad\qquad\qquad\qquad \text{if } e' < d \\ \\ x_d\alpha + (1 - x_d)((1 - \prod_{j=i+1}^{e'-1}(1 - x_j))\gamma\alpha \\ + \prod_{j=i+1}^{e'-1}(1 - x_j)\gamma\beta) \\ \qquad\qquad\qquad\qquad \text{if } e' = d; \end{cases}$$

  if n_i cheats, its expected welfare $EW_i^- = EP_i^-$ is

$$\begin{cases} (1 - \prod_{j=i+1}^{e'}(1 - x_j) \prod_{j=e'+1}^{d-1} x_j)\gamma\alpha \\ \qquad\qquad\qquad\qquad \text{if } e' < d \\ \\ x_d\alpha + (1 - x_d)(1 - \prod_{j=i+1}^{e'-1}(1 - x_j))\gamma\alpha \\ \qquad\qquad\qquad\qquad \text{if } e' = d. \end{cases}$$

  Therefore, we always have $EW_i^+ \geq EW_i^-$, which implies that telling the truth with probability 1 will bring the maximum expected welfare to n_i.

- Case C. $i = e'$. If n_i tells the truth, its expected welfare $EW_i^+ = EP_i^+$ is

$$\begin{cases} (1 - \prod_{j=i+1}^{d-1} x_j)\gamma\alpha + \prod_{j=i+1}^{d-1} x_j\gamma\beta \\ \qquad\qquad\qquad\qquad \text{if } e' < d \\ \\ \beta \qquad\qquad\qquad\qquad \text{if } e' = d; \end{cases}$$

if n_i cheats, its expected welfare $EW_i^- = EP_i^-$ is

$$\begin{cases} (1 - \prod_{j=i+1}^{d-1} x_j)\gamma\alpha \quad \text{if } e' < d \\ \\ 0 \qquad\qquad\qquad\qquad \text{if } e' = d. \end{cases}$$

As in the previous case, we always have $EW_i^+ \geq EW_i^-$, which implies that telling the truth with probability 1 will bring the maximum expected welfare to n_i.

- Case D. $e' < i \leq d$. Note that $T_d = FALSE$ here, which implies that n_i always tells the truth in case of $i = d$. So we only need to consider the case of $i < d$. If n_i tells the truth, the expected welfare is

$$EW_i^+ = EP_i^+ = (1 - \prod_{j=i+1}^{d-1} x_j)\gamma\alpha.$$

If n_i cheats, it gets an expected payment of

$$EP_i^- = (1 - \prod_{j=i+1}^{d-1} x_j)\gamma\alpha + \prod_{j=i+1}^{d-1} x_j\gamma\beta,$$

while its gets a cost of

$$U_i^- = \delta.$$

So its expected welfare is

$$EW_i^- = EP_i^- - U_i^- = (1 - \prod_{j=i+1}^{d-1} x_j)\gamma\alpha + \prod_{j=i+1}^{d-1} x_j\gamma\beta - \delta.$$

As in Cases B and C, we always have $EW_i^+ \geq EW_i^-$, which implies that telling the truth with probability 1 will bring the maximum expected welfare to n_i.

Besides individual cheating, we further consider the possibility of collusion.

Definition 39 *A game is* collusion-resistant*, if any group of colluding players cannot increase the expected sum of their welfare by using any strategy profile other than that in which everybody tells the truth.*

Theorem 40 *The receipt-submission game is collusion-resistant, if* $\delta \geq (d-1)\gamma\alpha$*, and* n_d *does not cheat in case of* $T_d = FALSE$*.*

Proof: Consider strategizing group G that uses a strategy profile other than everybody telling the truth. Suppose that each player $n_i \in G$ tells the truth with probability x_i. The expected sum of welfares of G is

$$EW_G = \sum_{L \subseteq G} \prod_{i \in L} (1 - x_i) \prod_{i \in G - L} x_i W_G(L),$$

where $W_G(L)$ denotes the sum of welfares of G in case that the set of lying players is L. Our goal is to show

$$EW_G \leq W_G(\phi).$$

Obviously, we only need to prove

$$\forall L \subseteq G, W_G(L) \leq W_G(\phi).$$

We distinguish two cases here. (Hereafter, we use $\sharp G(u,v)$ to denote $|\{i|u \leq i \leq v \wedge n_i \in G\}|$.)

- Case A. $n_0 \notin G$. By considering the indices of players in L, we further distinguish three subcases.

  - Subcase A-A. $\forall n_i \in L$, $i < e'$. Then trivially we have

$$W_G(L) = W_G(\phi).$$

  - Subcase A-B. $\forall n_i \in L$, $i \leq e'$, and $n_{e'} \in L$. Then $W_G(L)$ is equal to

$$\begin{cases} W_G(\phi) - ((e'-1)\gamma\alpha + \gamma\beta)) \\ \quad \text{if } e' < d \text{ and } \forall i \leq e',\, n_i \in L \\ \\ W_G(\phi) - ((e'-1)\alpha + \beta) \\ \quad \text{if } e' = d \text{ and } \forall i \leq e',\, n_i \in L \\ \\ W_G(\phi) - (\sharp G(e,e)(\gamma\alpha - \gamma\beta) \\ \quad + \sharp G(e+1, e'-1)\gamma\alpha + \gamma\beta) \\ \quad \text{if } e' < d \text{ and } \exists i \leq e', n_i \notin L \\ \\ W_G(\phi) - (\sharp G(1, e-1)(1-\gamma)\alpha \\ \quad + \sharp G(e,e)(\alpha - \gamma\beta) + \sharp G(e+1, e'-1)\alpha + \beta) \\ \quad \text{if } e' = d \text{ and } \exists i \leq e', n_i \notin L, \end{cases}$$

    where $e = \max_{i \leq e', n_i \notin L} i$. Therefore,

$$W_G(L) \leq W_G(\phi).$$

– Subcase A-C. $\exists n_i \in L$, $i > e'$. Then

$$W_G(L) = W_G(\phi) + \sharp G(e', e')(\gamma\alpha - \gamma\beta)$$
$$+\sharp G(e'+1, e-1)\gamma\alpha + \gamma\beta - \delta,$$

where $e = \max_{e'<i<d, n_i \in L} i$. It's easy to see

$$\begin{aligned} W_G(L) &\leq W_G(\phi) + (e - e')\gamma\alpha - \delta \\ &\leq W_G(\phi) + (d-1)\gamma\alpha - \delta \\ &\leq W_G(\phi). \end{aligned}$$

- Case B. $n_o \in G$. As in Case A, we further distinguish three subcases.

  – Subcase B-A. $\forall n_i \in L$, $i < e'$. Trivially, we have

$$W_G(L) = W_G(\phi).$$

– Subcase B-B. $\forall n_i \in L$, $i \leq e'$, and $n_{e'} \in L$. Then $W_G(L)$ is equal to

$$\begin{cases} W_G(\phi) + e'\gamma\beta - ((e'-1)\gamma\alpha + \gamma\beta)) \\ \quad \text{if } e' < d \text{ and } \forall i \leq e',\, n_i \in L \\ \\ W_G(\phi) + e'\gamma\beta - ((e'-1)\alpha + \beta) \\ \quad \text{if } e' = d \text{ and } \forall i \leq e',\, n_i \in L \\ \\ W_G(\phi) + (e'-e)\gamma\beta - (\sharp G(e,e)(\gamma\alpha - \gamma\beta) \\ \quad +\sharp G(e+1, e'-1)\gamma\alpha + \gamma\beta) \\ \quad \text{if } e' < d \text{ and } \exists i \leq e', n_i \notin L \\ \\ W_G(\phi) + (e'-e)\gamma\beta - (\sharp G(1, e-1) \\ \quad \cdot(1-\gamma)\alpha + \sharp G(e,e)(\alpha - \gamma\beta) \\ \quad +\sharp G(e+1, e'-1)\alpha + \beta) \\ \quad \text{if } e' = d \text{ and } \exists i \leq e', n_i \notin L, \end{cases}$$

where $e = \max_{i \leq e', n_i \notin L} i$. Because $\gamma\beta < \beta < \alpha$ and $\gamma\beta < \gamma\alpha$, it is easy to see

$$W_G(L) \leq W_G(\phi).$$

– Subcase B-C. $\exists n_i \in L$, $i > e'$. Then

$$\begin{aligned} W_G(L) &= W_G(\phi) - (e-e')\gamma\beta + \sharp G(e', e') \\ &\quad \cdot(\gamma\alpha - \gamma\beta) + \sharp G(e'+1, e-1)\gamma\alpha \\ &\quad +\gamma\beta - \delta, \end{aligned}$$

where $e = \max_{e'<i<d, n_i \in L} i$. It's easy to see

$$\begin{aligned} W_G(L) &\leq W_G(\phi) + (e - e')\gamma\alpha - \delta \\ &\leq W_G(\phi) + (d-1)\gamma\alpha - \delta \\ &\leq W_G(\phi). \end{aligned}$$

Definition 41 *A game is cheat-proof, if truth-telling is an optimal strategy for every player and the game is collusion-resistant.*

Theorem 42 *The receipt-submission game is cheat-proof, if* $\delta \geq \max(\gamma\beta, (d-1)\gamma\alpha)$, *and* n_d *does not cheat in case of* $T_d = FALSE$.

7.5.3 Incentive Analysis of Performance

In the above proofs, we have essentially shown that each selfish node should report faithfully to the CCS. With this knowledge in mind, comparing the expected gain of credit from forwarding a message with that of not forwarding the message, an intermediate node can expect a net gain of $p_2\alpha + (p_1 - p_2)\gamma\alpha + (1 - p_1)\gamma\beta - \gamma\beta$, where p_1 is the probability that the message arrives at the next node, and p_2 the probability that the message arrives at the destination. Simplifying, we have $p_2(1 - \gamma)\alpha + p_1\gamma(\alpha - \beta)$. Note that this payment gain is always greater than 0 because γ is small, and $\alpha > \beta$. If this payment gain is sufficient to cover the cost of forwarding a message, the node has incentive to forward the message.

7.6 Stimulating Cooperation in Route Discovery

Because route discovery uses broadcasts, the approach we have presented cannot be applied directly. In this section, we propose a slightly different approach, which is a bit more expensive. But because route discovery is performed less frequently, this approach is affordable in general. This approach is based on DSR, and essentially we will show how to improve DSR to stimulate cooperation in route discovery. Note that the reply to ROUTE REQUEST can be sent as a regular message. Therefore we only need to stimulate the re-broadcasting of ROUTE REQUEST.

7.6.1 Sending a ROUTE REQUEST

In general, when a node starts to broadcast a ROUTE REQUEST, the message includes the source address and a sequence number. Then the node signs and broadcasts the message, and increases its sequence number counter by 1.

7.6.2 Receiving a ROUTE REQUEST

Suppose that a node receives a ROUTE REQUEST. It first decides whether the message is a replay by looking at the sequence number. The node saves the received ROUTE REQUEST for getting payment in the future. When the node decides to rebroadcast the ROUTE REQUEST, it appends its own address to the ROUTE REQUEST and signs the extended message.

7.6.3 Computing Payment

When the CCS computes payment, a ROUTE REQUEST is rejected if any signature in the message is invalid. Furthermore, if a ROUTE REQUEST submitted by a node is a part of another ROUTE REQUEST submitted by the same node, then the former

message is rejected. Finally, the CCS builds a tree based on the accepted ROUTE REQUEST messages. The sender pays α to each non-leaf node of the tree, and β to each leaf of the tree. For each node outside the tree, the sender node pays $\alpha - \beta$ to the CCS.

7.7 Evaluations

7.7.1 Overhead

We first evaluate the overhead of our system. In order to measure the overhead, we have implemented a prototype of our system using the Crypto++4.0 library [38]. The implementation can run over a wide range of platforms such as Linux and Win32.

In the evaluations below, our mobile node is a Laptop with an *Intel Mobile Pentium III* processor at 866MHz. The OS of the mobile node is Windows XP. The length of a message payload is 1000 bytes. The message digest function is MD5. We consider two digital signature schemes: RSA with a modulus of 1024 bits, and ECNR over GF(p) 168 [77]. We assume that the average path length is 8 hops.

	send (ms)	forward (ms)	authentication header (bytes)	receipt (bytes)
RSA 1024	10.4	0.3	128	180
ECNR over GF(p) 168	7.3	13.2	42	94
ECNR over GF(p) 168 (precomputation)	3.7	6.1	42	94

Table 7.1: CPU processing time; sizes of authentication header and receipts.

We first evaluate the CPU processing time on a mobile node. In our system, the major online processing overhead is the signing operation by the sender and the verification operation by the intermediate nodes. The second and third columns of Table I show the CPU processing time of the sender to send a message and that of

an intermediate node to forward a message, respectively. We observe that RSA has a much smaller forwarding overhead. Thus, if reducing forwarding overhead is the major objective, RSA is a better implementation choice. However, for both schemes, we observe that the CPU processing time is acceptable, if the nodes do not send a large number of messages, which is the expected case when the mobile nodes have limited bandwidth and energy.

We next evaluate the bandwidth and storage requirement. Compared with a message using DSR as the routing protocol but without message authentication, the major increased overhead is the digital signature for message authentication. For RSA with a modulus of 1024 bits, the authentication header is about 128 bytes; for ECNR GF(p) with 168 bits, the header is about 42 bytes. In terms of storage requirement for the receipts, for RSA 1024, the total storage of a receipt is 180 bytes, and for the Elliptic Curve based ECNR, it is 94 bytes. Comparing RSA with ECNR, we observe that ECNR has a much smaller bandwidth and storage requirement.

7.7.2 System Performance vs. Network Resources

We next evaluate the performance of our system. One major metric of the performance of our system is the message success rate, *i.e.*, the percentage of messages that are successfully relayed from the sender to the destination. For the purpose of this evaluation, we ignore message drops due to channel errors.

We first note that this success rate will depend on the sending/forwarding strategy of the mobile nodes. As we have discussed in Section 7.3, although our system provides incentive for cooperation by giving more credit for forwarding a message, whether or not to forward a specific message will depend on the objectives and the status of a node. To demonstrate the generality of our system, for the purpose of this evaluation, we consider a special class of mobile nodes, namely the power-and-

credit-conservative nodes. Specifically, a node is power-conservative if its remaining power allows it to send (and forward) only a limited amount of messages; a node is credit-conservative if it refrains from sending any new message when its credit balance is insufficient to cover the charge for sending a message. For this type of nodes, we consider the following strategy: when it receives a transient message, if the number of messages allowed to be sent by its estimated credit balance is smaller than the number of messages allowed to be sent by its remaining battery, forward the transient message and increase its estimated credit balance by $p\alpha$, where p is the probability that the forwarded message will arrive at its destination; otherwise, drop the message. In summary, let c and b denote the estimated credit balance and the number of messages allowed to be transmitted by the remaining battery of a node, respectively. Assume that each message takes an average of L hops. Then the policy of such a node is the following: if $\frac{c}{L} < b$, forward a transient message; otherwise, drop the message. Given the strategy above, we next evaluate the message success rate of our system.

We first evaluate the message success rate under different configurations of network resources. Figure 7.7 shows the message success rates for two ad hoc networks: one network with 70 nodes uniformly distributed in an area of 1000 by 1000, and another network with 200 nodes uniformly distributed in an area of 2000 by 2000. The communication radius of each node is 250. In this experiment, because the nodes are power-and-credit-conservative, their estimated credit balance c is close to 0 and we choose their initial credit to be uniformly distributed in $[0, C]$, where $C = 10$. To observe the effect of the amount of node resource on the overall message success rate, for each node, we choose its b, the number of messages that can be sent/forwarded by the remaining battery of the node, uniformly from $[0, B]$, where B is from 30 to 640. Note that even the maximum number of 640 is very conservative [137]. To

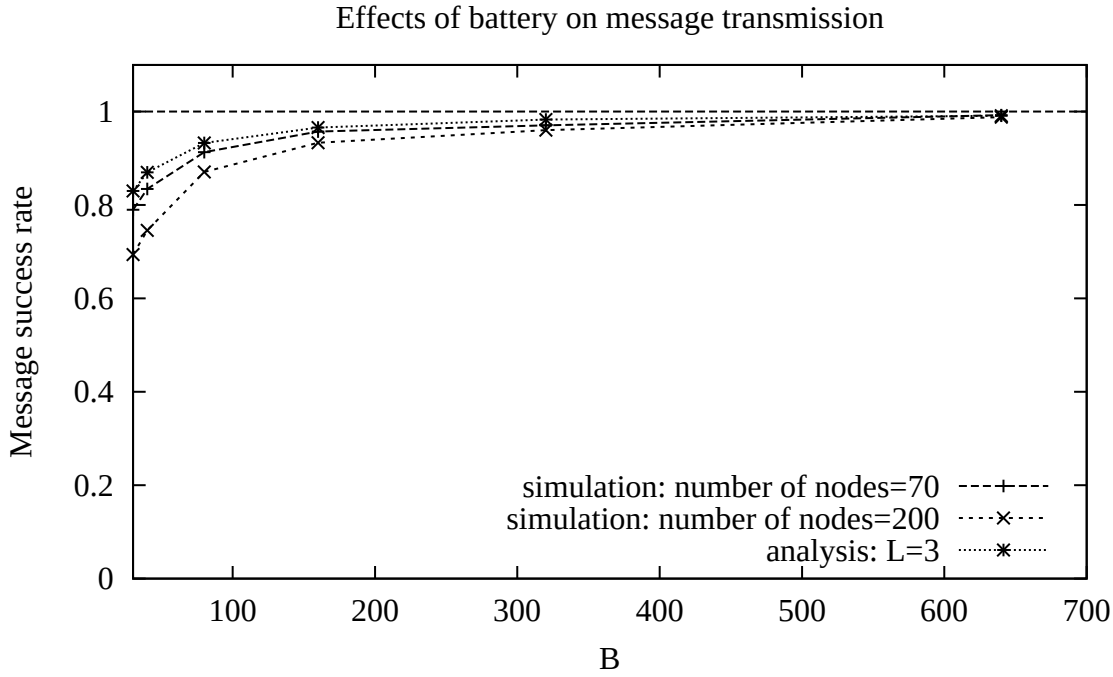

Figure 7.7: Message success rate vs. network battery resources.

control the number of experiments for each configuration, we repeat the experiment of a configuration with a different random seed until the 5% confidence interval is narrower than 5% of the mean value. From Figure 7.7, we observe clearly that with increasing resources, the nodes are more willing to forward others' messages, and therefore the message success rate is very close to 1.

We next evaluate the dynamics of message success rate; that is, how message success rate evolves as the nodes send more messages. Figure 7.8 shows the result. Under this experiment, the initial credit of each node is 3, and the initial battery of each node is B, where $B = 100$ or 500. The value of $B = 100$ is in the very low end, and the objective is to observe message drops. The x-axis of Figure 7.8 is the index of the number of messages generated by the mobile nodes, and the y-axis shows the message success rate. From this figure, we observe that as system evolves and no new node joins, the batteries of the nodes are consumed and the nodes tend to be

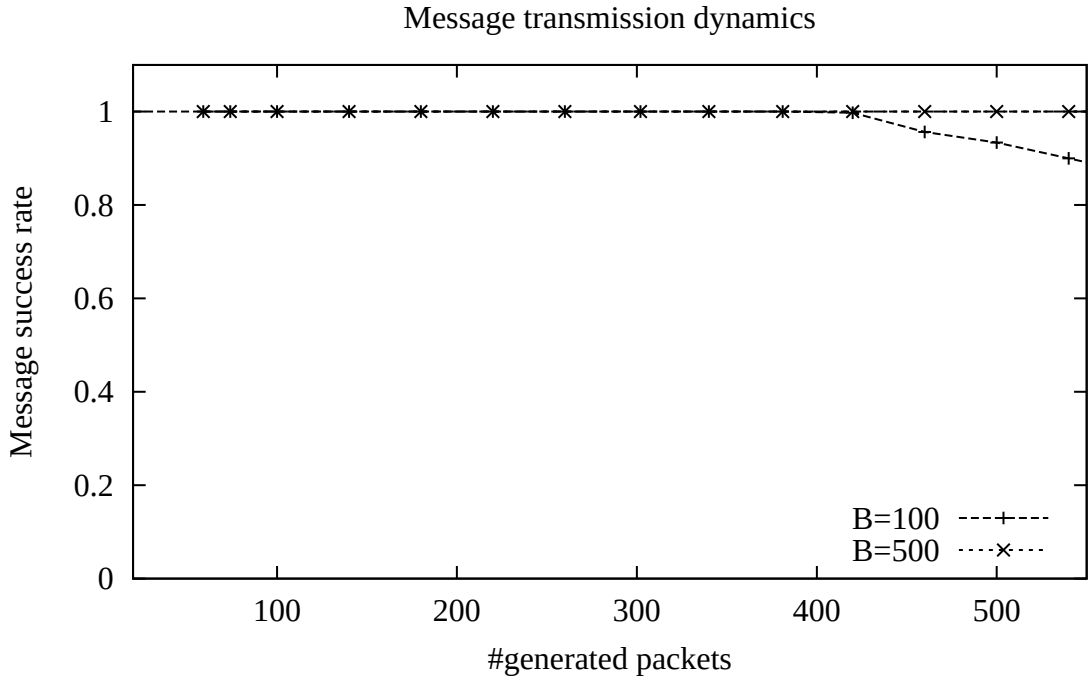

Figure 7.8: Dynamics of message success rate.

more conservative. However, we observe that, even in a low battery configuration, considerable number of messages will be generated before the message success rate decreases.

7.8 Summary of the Work on Incentives in Mobile Ad hoc Networks

In this chapter, we present Sprite, a system to provide incentive to mobile nodes to cooperate. Our system determines payments and charges from a game-theoretic perspective, and we show that our system motivates each node to report its behavior honestly, even when a collection of the selfish nodes collude. We also model the essential component of our system as the receipt-submission game, and prove the correctness of our system under this model. Our main result works for packet-forwarding,

and we extend it to route discovery as well. We also implement a prototype of our system and show the overhead of our system is insignificant. Simulations and analysis of the power-and-credit-conservative nodes show that the nodes can cooperate and forward each other's messages, unless the resources of the nodes are extremely low.

Chapter 8

Conclusion

In this dissertation, I have studied privacy, integrity, and incentive compatibility in computation with untrusted parties. All these are important security issues that need to be addressed in various scenarios, from different perspectives, and with high efficiency. As mentioned in Chapter 1, the problems of privacy and integrity belong to more traditional research of secure multi-party computation, while the study of incentive compatibility is a natural extension of secure multi-party computation to selfish, or economically rational, parties.

The technical contents of this dissertation can be divided into three parts: Chapter 2 presents definitions and frequently used techniques; Chapters 3 through 6 solve practical problems of privacy and integrity in various scenarios, using the techniques from Chapter 2 plus some other techniques specifically designed for these problems; Chapter 7 is a first attempt to add incentive considerations to multi-party computation problems. On the one hand, the work presented in Chapters 3 through 7 is an illustration of the general methodology of applying mathematical (*e.g.*, cryptographic and game theoretic) techniques to solve the security problems that arise in practice. On the other hand, the concrete problems studied are of significant impor-

tance on their own in practice. I hope that this dissertation will be a good start of my research on information and network security.

Bibliography

[1] M. Abe. Universally Verifiable Mix-Net with Verification Work Independent of the Number of Mix-Servers. In *Advances in Cryptology - Proceedings of EUROCRYPT 98*, volume 1403 of *Lecture Notes in Computer Science*, pp. 437–447. Springer-Verlag, 1998.

[2] M. Abe. Mix-Networks on Permutation Networks. In *Advances in Cryptology - - ASIACRYPT '99*, volume 1706 of *Lecture Notes in Computer Science*, pp. 258–273. Springer-Verlag, 1999.

[3] M. Abe and F. Hoshino. Remarks on Mix-Networks based on Permutation Networks. In *Proceedings of PKC'01*, volume 1992 of *Lecture Notes in Computer Science*, pp. 317–324. Springer-Verlag, 2001.

[4] D. Agrawal and C. C. Agrawal. On the Design and Quantification of Privacy Preserving Data Mining Algorithms. In *Proceedings of 12th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pp. 247–255. ACM, 2001.

[5] D. Agrawal and R. Srikant. Privacy Preserving Mining. In *Proceedings of the 2000 ACM SIGMOD Conference on Management of Data*, pp. 439–450. ACM, 2000.

[6] R. Agrawal and R. Srikant. Fast Algorithm for Mining Association Rules. In *Proceedings of 20th International Conference on Very Large Data Bases*, pp. 487–499. Morgan Kaufmann, 1994.

[7] W. Aiello, Y. Ishai, and O. Reingold. Priced Oblivious Transfer: How to Sell Digital Goods. In *Advances in Cryptology - Proceedings of EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pp. 119–135. Springer-Verlag, 2001.

[8] J. Algesheimer, C. Cachin, J. Camenisch, and G. Karjoth. Cryptographic Security for Mobile Code. In *IEEE Symposium on Security and Privacy*, pp. 2–11. IEEE, 2001.

[9] R. J. Anderson. The Eternity Service. In *Proceedings of Pragocrypt 96*, pp. 242–252. CTU, 1996.

[10] J. Aspnes, J. Feigenbaum, A. Yampolskiy, and S. Zhong. Towards a Theory of Entanglement. In *Proceedings of ESORICS'04*, Lecture Notes in Computer Science. Springer-Verlag, to appear.

[11] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, and K. Yang. On the (Im)possibility of Obfuscating Programs. In *Advances in Cryptology - Proceedings of CRYPTO 2001*, pp. 1–18. Springer-Verlag, 2001.

[12] D. Beaver, S. Micali, and P. Rogaway. The Round Complexity of Secure Protocols. In *Proceedings of the 22th Annual ACM Symposium on the Theory of Computing*, pp. 503–513. ACM, 1990.

[13] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations Among Notions of Security for Public-Key Encryption Schemes. In *Advances in Cryptology - Proceedings of CRYPTO 98*, pp. 26–45. Springer-Verlag, 1998.

[14] M. Bellare and S. Micali. Non-Interactive Oblivious Transfer and Applications. In *Advances in Cryptology - Proceedings of CRYPTO 89*, volume 435 of *Lecture Notes in Computer Science*, pp. 547–557. Springer-Verlag, 1990.

[15] M. Bellare and P. Rogaway. Random Oracles Are Practical: a Paradigm for Designing Efficient Protocols. In *Proc. First Annual Conference on Computer and Communications Security*, pp. 62–73. ACM, 1993.

[16] M. Ben-Or, S. Goldwasser, and A. Widgerson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation. In *Proceedings of the 20th Annual ACM Symposium on the Theory of Computing*, pp. 1–10. ACM, 1988.

[17] N. Ben Salem, L. Buttyan, J. P. Hubaux, and M. Jakobsson. A Charging and Rewarding Scheme for Packet Forwarding in Multi-hop Cellular Networks. In *Proceedings of MobiHoc'03*, pp. 13–24. ACM, 2003.

[18] D. Boneh. The Decision Diffie-Hellman Problem. In *ANTS-III*, volume 1423 of *Lecture Notes in Computer Science*, pp. 48–63. Springer-Verlag, 1998.

[19] D. Boneh and M. Franklin. Efficient Generation of Shared RSA Keys. In *Advances in Cryptology - Proceedings of CRYPTO 97*, volume 1294 of *Lecture Notes in Computer Science*, pp. 425–439. Springer-Verlag, 1997.

[20] G. Brassard, C. Crepeau, and J.-M. Robert. All-or-Nothing Disclosure of Secrets. In *Advances in Cryptology - Proceedings of CRYPTO 86*, volume 263 of *Lecture Notes in Computer Science*, pp. 234–238. Springer-Verlag, 1986.

[21] S. Buchegger and J.-Y. L. Boudec. Nodes Bearing Grudges: Towards Routing Security, Fairness, and Robustness in Mobile Ad Hoc Networks. In *10th Eu-*

romicro Workshop on Parallel, Distributed and Network-based Processing, pp. 403–410. IEEE, 2002.

[22] S. Buchegger and J.-Y. L. Boudec. Performance Analysis of the CONFIDANT Protocol: Cooperation Of Nodes - Fairness In Dynamic Ad-hoc NeTworks. In *Proceedings of IEEE/ACM Workshop on Mobile Ad Hoc Networking and Computing (MobiHOC)*, pp. 226–236. IEEE, 2002.

[23] L. Buttyan and J. P. Hubaux. Enforcing Service Availability in Mobile Ad-Hoc WANs. In *IEEE/ACM Workshop on Mobile Ad Hoc Networking and Computing (MobiHOC)*, pp. 87–96. ACM, 2000.

[24] L. Buttyan and J. P. Hubaux. Stimulating Cooperation in Self-Organizing Mobile Ad Hoc Networks. *Mobile Networks and Applications (MONET), special issue on Mobile Ad Hoc Networks*, **8**(2003), pp. 579–592.

[25] C. Cachin, J. Camenisch, J. Kilian, and J. Müller. One-Round Secure Computation and Secure Autonomous Mobile Agents. In *Automata, Languages and Programming, 27th International Colloquium,*, volume 1853 of *Lecture Notes in Computer Science*, pp. 512–523. Springer-Verlag, 2000.

[26] R. Canetti, Y. Dodis, S. Halevi, E. Kushilevitz, and A. Sahai. Exposure-Resilient Functions and All-or-Nothing Transforms. In *Advances in Cryptology - Proceedings of EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pp. 453–469. Springer-Verlag, 2000.

[27] M. Castro and B. Liskov. Practical Byzantine Fault Tolerance. In *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation*, pp. 173–186. ACM, 1999.

[28] D. Chaum. Untraceable Electronic Mail, Return Address and Digital Pseudonyms. *Communications of the ACM*, **24**(1981), pp. 84–88.

[29] D. Chaum, C. Crépeau, and I. Damgård. Multiparty Unconditionally Secure Protocols. In *Proceedings of the 20th Annual ACM Symposium on the Theory of Computing*, pp. 11–19. ACM, 1988.

[30] D. Chaum and T. Pedersen. Wallet Databases with Observers. In *Advances in Cryptology - Proceedings of CRYPTO 92*, volume 740 of *Lecture Notes in Computer Science*, pp. 89–105. Springer-Verlag, 1993.

[31] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private Information Retrieval. *Journal of the ACM*, **45**(1998), pp. 965–982.

[32] I. Clarke, O. Sandberg, B. Wiley, and T. Hong. Freenet: a Distributed Information Storage and Retrieval System. In *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability*, volume 2009 of *Lecture Notes in Computer Science*, pp. 46–66. Springer-Verlag, 2000.

[33] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols. In *Advances in Cryptology - Proceedings of CRYPTO 94*, volume 839 of *Lecture Notes in Computer Science*, pp. 174–187. Springer-Verlag, 1994.

[34] R. Cramer and I. Damgård. Secure Distributed Linear Algebra in a Constant Number of Rounds. In *Advances in Cryptology - Proceedings of CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pp. 119–136. Springer-Verlag, 2001.

[35] R. Cramer, I. Damgård, and J. B. Nielsen. Multiparty Computation from Threshold Homomorphic Encryption. In *Advances in Cryptology - Proceedings of EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pp. 280–299. Springer-Verlag, 2001.

[36] G. D. Crescenzo, T. Malkin, and R. Ostrovsky. Single Database Private Information Retrieval Implies Oblivious Transfer. In *Advances in Cryptology - Proceedings of EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pp. 122–138. Springer-Verlag, 2000.

[37] B. Dahill, B. N. Levine, E. Royer, and C. Shields. A Secure Routing Protocol for Ad Hoc Networks. Technical report, University of Massachusetts at Amherst, 2001.

[38] W. Dai. Crypto++4.0. Available at http://www.eskimo.com/ weidai/cryptlib.html.

[39] E. Dasseni, V. S. Verykios, A. K. Elmagarmid, and E. Bertino. Hiding Association Rules by Using Confidence and Support. In *Information Hiding Workshop*, volume 2137 of *Lecture Notes in Computer Science*, pp. 369–383. Springer-Verlag, 2001.

[40] Y. Desmedt and Y. Frankel. Threshold Cryptosystems. In *Advances in Cryptology - Proceedings of CRYPTO 89*, volume 435 of *Lecture Notes in Computer Science*, pp. 307–315. Springer-Verlag, 1990.

[41] W. Du and M. J. Atallah. Secure Multi-Party Computation Problems and their Applications: a Review and Open Problems. In *New Security Paradigms Workshop*, pp. 11–20. ACM, 2001.

[42] S. Even, O. Goldreich, and A. Lempel. A Randomized Protocol for Signing Contracts. *Communications of the ACM*, **28**(1985), pp. 637–647.

[43] A. Evfimievski, R. Srikant, D. Agrawal, and J. Gehrke. Privacy Preserving Mining of Association Rules. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 217–228. ACM, 2002.

[44] J. Feigenbaum, C. Papadimitriou, R. Sami, and S. Shenker. A BGP-based Mechanism for Lowest-Cost Routing. In *Proceedings of the 2002 ACM Symposium on Principles of Distributed Computing.*, pp. 173–182. ACM, 2002.

[45] J. Feigenbaum, C. Papadimitriou, and S. Shenker. Sharing the Cost of Multicast Transmissions. *Journal of Computer and System Sciences (Special issue on Internet Algorithms.)*, **63**(2001), pp. 21–41.

[46] A. Fiat and A. Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Advances in Cryptology - Proceedings of CRYPTO 86*, volume 263 of *Lecture Notes in Computer Science*, pp. 186–194. Springer-Verlag, 1987.

[47] M. Franklin and M. Yung. Communiation Complexity of Secure Computation. In *Proceedings of the 24th Annual ACM Symposium on the Theory of Computing*, pp. 699–710. ACM, 1992.

[48] K. Fu, F. Kaashoek, and D. Mazieres. Fast and Secure Distributed Read-Only File System. In *Proceedings of the 4th Symposium on Operating Systems Design and Implementation*, pp. 181–196, 2000.

[49] J. Furukawa, H. Miyauchi, K. Mori, S. Obana, and K. Sako. An Implementation of a Universally Verifiable Electronic Voting Scheme based on Shuffling.

In *Financial Cryptography 2002*, volume 2357 of *Lecture Notes in Computer Science*, pp. 16–30. Springer-Verlag, 2002.

[50] J. Furukawa and K. Sako. An Efficient Scheme for Proving a Shuffle. In *Advances in Cryptology - Proceedings of CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pp. 368–387. Springer-Verlag, 2001.

[51] E. Gabber, P. Gibbons, Y. Matias, and A. Mayer. How to Make Personalized Web Browsing Simple, Secure and Anonymous. In *Proceedings of Financial Cryptography'97*, pp. 17–31. Springer-Verlag, 1997.

[52] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure Distributed Key Generation for Discrete-Log Based Cryptosystems. In *Advances in Cryptology - Proceedings of EUROCRYPT 99*, volume 1592 of *Lecture Notes in Computer Science*, pp. 295–310. Springer-Verlag, 1999.

[53] R. Gennaro, M. Rabin, and T. Rabin. Simplified VSS and Fast-track Multiparty Computations with Applications to Threshold Cryptography. In *PODC: 17th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, pp. 101–111. ACM, 1998.

[54] Y. Gertner, S. Goldwasser, and T. Malkin. A Random Server Model for Private Information Retrieval. In *RANDOM'98*, volume 1518 of *Lecture Notes in Computer Science*, pp. 200–217. Springer-Verlag, 1998.

[55] Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin. Protecting Data Privacy in Private Information Retrieval Schemes. In *Proceedings of the 30th Annual ACM Symposium on the Theory of Computing*, pp. 151–160. ACM, 1998.

[56] G. A. Gibson, D. F. Nagle, K. Amiri, J. Butler, F. W. Chang, H. Gobioff,

C. Hardin, E. Riedel, D. Rochberg, and J. Zelenka. A Cost-Effective, High-Bandwidth Storage Architecture. In *Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 92–103. ACM, 1998.

[57] E. Goh, H. Shacham, N. Mdadugu, and D. Boneh. SiRiUS: Securing Remote Untrusted Storage. In *Proceedings of the ISOC Network and Distributed System Security Symposium (NDSS)*, pp. 131–145. ISOC, 2003.

[58] A. Goldberg and P. Yianilos. Towards an Archival Intermemory. In *Proceedings of the IEEE International Forum on Research and Technology, Advances in Digital Libraries (ADL '98)*, pp. 147–156. IEEE, 1998.

[59] O. Goldreich. Secure Multi-Party Computation. Working Draft Version 1.1, 1998.

[60] O. Goldreich, S. Micali, and A. Wigderson. How to Play ANY Mental Game. In *Proceedings of the 19th Annual ACM Symposium on the Theory of Computing*, pp. 218–229. ACM, 1987.

[61] O. Goldreich, S. Micali, and A. Wigderson. Proofs that Yield Nothing but Their Validity or All Languages in NP Have Zero-Knowledge Proof Systems. *Journal of the Association for Computing Machinery*, **38**(1991), pp. 691–729.

[62] S. Goldwasser. Multi-party Computations: Past and Present. In *Proceedings of the sixteenth annual ACM symposium on Principles of Distributed Computing*, pp. 1–6. ACM, 1997.

[63] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, **28**(1984), pp. 270–299.

[64] S. Goldwasser, S. Micali, and R. Rivest. A Digital Signature Scheme Secure against Adaptive Chosen Message Attack. *SIAM Journal on Computing*, **17**(1988), pp. 281–308.

[65] P. Golle, K. Leyton-Brown, and I. Mironov. Incentives in Peer-to-Peer File Sharing. In *Proceedings of the ACM Symposium on Electronic Commerce 2001 (EC' 01)*, pp. 264–267. ACM, 2001.

[66] P. Golle, S. Zhong, D. Boneh, M. Jakobsson, and A. Juels. Optimistic Mixing for Exit-Polls. In *Advances in Cryptology - ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pp. 451–465. Springer-Verlag, 2002.

[67] L. Gong. Java Security Architecture (JDK1.2). Technical report, Sun Microsystems, 1998.

[68] J. Hershberger and S. Suri. Vickrey Prices and Shortest Paths: What is an Edge Worth. In *Proceedings of the 42nd Annual Symposium on Foudations of Computer Science 2001*, pp. 252–259. IEEE, 2001.

[69] M. Hirt and U. Maurer. Robustness for Free in Unconditional Multi-party Computation. In *Advances in Cryptology - Proceedings of CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pp. 101–118. Springer-Verlag, 2001.

[70] M. Hirt, U. Maurer, and B. Prztdatek. Efficient Secure Multi-party Computation. In *Advances in Cryptology - ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pp. 143–161. Springer-Verlag, 2000.

[71] M. Hirt and K. Sako. Efficient Receipt-Free Voting Based on Homomorphic Encryption. In *Advances in Cryptology - Proceedings of EUROCRYPT 2000*,

volume 1807 of *Lecture Notes in Computer Science*, pp. 539–556. Springer-Verlag, 2000.

[72] H.-Y. Hsieh and R. Sivakumar. Performance Comparison of Cellular and Multi-hop Wireless Networks: A Quantitative Study. In *Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS) 2001*, pp. 113–122. ACM, 2001.

[73] Y.-C. Hu, A. Perrig, and D. B. Johnson. Ariadne: A Secure On-Demand Routing Protocol for Ad Hoc Networks. Technical Report TR01-384, Department of Computer Science, Rice University, December 2001.

[74] J. P. Hubaux, J. Y. L. Boudec, S. Giordano, M. Hamdi, L. Blazevic, L. Buttyan, and M. Vojnovic. Towards Mobile Ad-Hoc WANs: Terminodes. In *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 1052–1059. IEEE, 2000.

[75] J. P. Hubaux, L. Buttyan, and S. Capkun. The Quest for Security in Mobile Ad Hoc Networks. In *Proceedings of ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHOC)*, pp. 146–155. ACM, October 2001.

[76] J. P. Hubaux, T. Gross, J. Y. L. Boudec, and M. Vetterli. Towards self-organized mobile ad hoc networks: the Terminodes project. *IEEE Communications Magazine*, **39**(2001), pp. 118–124.

[77] IEEE P1363 Group. IEEE P1363 Standard. Available at http://grouper.ieee.org/groups/1363/index.html.

[78] M. Jakobsson. A Practical Mix. In *Advances in Cryptology - Proceedings of EUROCRYPT 98*, volume 1403 of *Lecture Notes in Computer Science*, pp. 448–461. Springer-Verlag, 1998.

[79] M. Jakobsson. Flash Mixing. In *Proceedings of the Eighteenth Annual ACM Symposium on Principles of Distributed Computing*, pp. 83–89. ACM, 1999.

[80] M. Jakobsson, J. P. Hubaux, and L. Buttyan. A Micropayment Scheme Encouraging Collaboration in Multi-Hop Cellular Networks. In *Proceedings of Financial Crypto 2003*, volume 2742 of *Lecture Notes in Computer Science*, pp. 15–33. Springer-Verlag, 2003.

[81] M. Jakobsson and A. Juels. Millimix: Mixing in Small Batches. Technical Report 99-33, DIMACS, 1999.

[82] M. Jakobsson and A. Juels. An Optimally Robust Hybrid Mix Network. In *Proceedings of the Twentieth Annual ACM Symposium on Principles of Distributed Computing*, pp. 284–292. ACM, 2001.

[83] M. Jakobsson and D. M. Rayhi. Mix-based Electronic Payments. In *Proceedings of SAC98*, volume 1556 of *Lecture Notes in Computer Science*, pp. 157–173. Springer-Verlag, 1998.

[84] M. Jakobsson, S. Wetzel, and B. Yener. Stealth Attacks on Ad Hoc Wireless Networks. In *Proceedings of IEEE VTC'03*, pp. 2103–2111. IEEE, 2003.

[85] D. B. Johnson and D. A. Malt. *Mobile Computing (Tomasz Imielinski and Hank Korth, eds.)*, chapter Dynamic Source Routing in Ad Hoc Wireless Networks. Kluwer Academic Publishers, 1996.

[86] M. Kantarcioglu and C. Clifton. Privacy Preserving Distributed Mining of Association Rules on Horizontally Partitioned Data. In *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pp. 639–644. ACM, 2002.

[87] M. Kantarcioglu and C. Clifton. Privacy Preserving Distributed Mining of Association Rules on Horizontally Partitioned Data. *IEEE Transactions of Knowledge and Data Engineering*, (to appear).

[88] A. Kiayias and M. Yung. Self-Tallying Elections and Perfect Ballot Secrecy. In *Proc. of PKC'02*, volume 2274 of *Lecture Notes in Computer Science*, pp. 141–158. Springer-Verlag, 2002.

[89] E. Kushilevitz and R. Ostrovsky. Replication is NOT Needed: SINGLE Database, Computationally-Private Information Retrieval. In *Proceedings of the 38th IEEE Symposium on Foundations of Computer Science*, pp. 364–373. IEEE, 1997.

[90] Y. Lindell and B. Pinkas. Privacy Preserving Data Mining. In *Advances in Cryptology - Proceedings of CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pp. 36–54. Springer-Verlag, 2000.

[91] H. Luo, P. Zerfos, J. Kong, S. Lu, and L. Zhang. Self-Securing Ad-Hoc Wireless Networks. In *IEEE Symposium on Computers and Communications*, pp. 567–574. IEEE, 2002.

[92] U. Maheshwari and R. Vingralek. How to Build a Trusted Database System on Untrusted Storage. In *Proceedings of the 4th Symposium on Operating Systems Design and Implementation*, pp. 135–150. ACM, 2000.

[93] S. Marti, T. Giuli, K. Lai, and M. Baker. Mitigating Routing Misbehavior in Mobile Ad Hoc Networks. In *Proceedings of The Seventh International Conference on Mobile Computing and Networking*, pp. 255–265. ACM, 2000.

[94] D. Mazieres and D. Shasha. Don't Trust Your File Server. In *Proceedings of the*

8th IEEE Workshop on Hot Topics in Operating Systems, pp. 99–104. IEEE, 2001.

[95] D. Mazieres and D. Shasha. Building Secure File Systems out of Byantine Storage. In *Proceedings of the Twenty-First Annual ACM Symposium on Principles of Distributed Computing*, pp. 108–117. ACM, 2002.

[96] D. Mazieres and M. Waldman. Tangler - a censorship-resistant publishing system based on document entanglements. In *Proceedings of the 8th ACM Conference on Computer and Communications Security*, pp. 126–135. ACM, 2001.

[97] R. Merkle. Protocols for Public Key Cryptosystems. In *IEEE Symposium on Security and Privacy*, pp. 122–134. IEEE, 1980.

[98] M. Mitomo and K. Kurosawa. Attack for Flash Mix. In *Advances in Cryptology - ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pp. 192–204. Springer-Verlag, 2000.

[99] Mojo Nation. Technology Overview, 2000.

[100] M. Naor and B. Pinkas. Oblivious Transfer and Polynomial Evaluation. In *Proceedings of the 31st Annual ACM Symposium on the Theory of Computing*, pp. 245–254. ACM, 1999.

[101] M. Naor and B. Pinkas. Oblivious Transfer with Adaptive Queries. In *Advances in Cryptology - Proceedings of CRYPTO 99*, volume 1666 of *Lecture Notes in Computer Science*, pp. 573–590. Springer-Verlag, 1999.

[102] M. Naor and B. Pinkas. Distributed Oblivious Transfer. In *Advances in Cryp-*

tology - ASIACRYPT 2000, volume 1976 of *Lecture Notes in Computer Science*, pp. 205–219. Springer-Verlag, 2000.

[103] G. C. Necula and P. Lee. Safe, Untrusted Agents Using Proof-Carrying Code. In *Mobile Agents and Security*, volume 1419 of *Lecture Notes in Computer Science*, pp. 61–91. Springer-Verlag, 1998.

[104] A. Neff. A Verifiable Secret Shuffle and its Application to E-Voting. In *Proceedings of the 8th ACM Conference on Computer and Communications Security*, pp. 116–125. ACM, 2001.

[105] N. Nisan. Algorithms for Selfish Agents. In *16th Annual Symposium on Theoretical Aspects of Computer Science*, volume 1563 of *Lecture Notes in Computer Science*, pp. 1–15. Springer-Verlag, 1999.

[106] N. Nisan and A. Ronen. Algorithmic Mechanism Design. *Games and Economic Behavior*, **35**(2001), pp. 166–196.

[107] NIST. Secure hash standard. Federal Information Processing Standards Publication 180-1, 1995.

[108] W. Ogata, K. Kurosawa, K. Sako, and K. Takatani. Fault Tolerant Anonymous Channel. In *Information and Communication Security, First International Conference*, volume 1334 of *Lecture Notes in Computer Science*, pp. 440–444. Springer-Verlag, 1997.

[109] S. R. M. Oliveira and O. R. Zaiane. Privacy Preserving Frequent Itemset Mining. In *IEEE International Conference on Data Mining Workshop on Privacy, Security and Data Mining*, volume 14, pp. 43–54. IEEE, 2002.

[110] M. J. Osborne and A. Rubenstein. *A Course in Game Theory*. The MIT Press, 1994.

[111] P. Paillier. Public-Key Cryptosystems based on Composite Degree Residuosity Classes. In *Advances in Cryptology - Proceedings of EUROCRYPT 99*, volume 1592 of *Lecture Notes in Computer Science*, pp. 223–238. Springer-Verlag, 1999.

[112] C. H. Papadimitriou. Algorithms, Games, and the Internet. In *Proceedings of the 33th Annual ACM Symposium on the Theory of Computing*, pp. 749–753. ACM, 2001.

[113] C. Park, K. Itoh, and K. Kurosawa. Efficient Anonymous Channel and All/Nothing Election Scheme. In *Advances in Cryptology - Proceedings of EUROCRYPT 93*, volume 765 of *LNCS*, pp. 248–259. Springer-Verlag, 1993.

[114] T. Pedersen. A Threshold Cryptosystem without a Trusted Third Party. In *Advances in Cryptology - Proceedings of EUROCRYPT 91*, volume 547 of *Lecture Notes in Computer Science*, pp. 522–526. Springer-Verlag, 1991.

[115] C. Perkins. *Ad Hoc Networking*. Addison-Wesley, 2000.

[116] B. Pfitzmann. Breaking an Efficient Anonymous Channel. In *Advances in Cryptology - Proceedings of EUROCRYPT 94*, volume 950 of *Lecture Notes in Computer Science*, pp. 339–348. Springer-Verlag, 1995.

[117] B. Pfitzmann and A. Pfitzmann. How to Break the Direct RSA-Implementation of Mixes. In *Advances in Cryptology - Proceedings of EUROCRYPT 89*, volume 434 of *Lecture Notes in Computer Science*, pp. 373–381. Springer-Verlag, 1989.

[118] Y. Qiu and P. Marbach. Bandwidth Allocation in Wireless Ad Hoc Networks:

A Price-Based Approach. In *Proceedings of IEEE INFOCOM 2003*, pp. 797–807. IEEE, 2003.

[119] M. Rabin. How to Exchange Secrets by Oblivious Transfer. Technical Report TR-81, Aiken Computation Laboratory, Harvard University, 1981.

[120] R. Rivest. The MD5 Message-Digest Algorithm. IETF NetworkWorking Group, RFC 1321, 1992.

[121] R. Rivest. All-or-Nothing Encryption and the Package Transform. In *Fast Software Encryption*, volume 1267 of *Lecture Notes in Computer Science*, pp. 210–218. Springer-Verlag, 1997.

[122] S. J. Rizvi and J. R. Haritsa. Maintaining Data Privacy in Association Rule Mining. In *Proceedings of 28th International Conference on Very Large Data Bases*, pp. 682–693. Morgan Kaufmann, 2002.

[123] T. Roughgarden and E. Tardos. How bad is selfish routing? *Journal of the Association for Computing Machinery*, **49**(2002), pp. 236–259.

[124] K. Sako and J. Kilian. Receipt-Free Mix-Type Voting Schemes — a Practical Solution to the Implementation of a Voting Booth. In *Advances in Cryptology - Proceedings of EUROCRYPT 95*, volume 921 of *Lecture Notes in Computer Science*, pp. 393–403. Springer-Verlag, 1995.

[125] T. Sander and C. Tschudin. Protecting Mobile Agents against Malicious Hosts. In *Mobile Agents and Security*, volume 1419 of *Lecture Notes in Computer Science*, pp. 44–60. Springer-Verlag, 1998.

[126] T. Sander, A. Young, and M. Yung. Non-Interactive CryptoComputing for

NC^1. In *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science*, pp. 554–567. IEEE, 1998.

[127] S. Savage, N. Cardwell, D. Wetherall, and T. Anderson. TCP Congestion Control with a Misbehaving Receiver. *ACM Computer Communication Review*, **29**(1999), pp. 71–78.

[128] Y. Saygin, V. S. Verykios, and A. K. Elmagarmid. Privacy Preserving Association Rule Mining. In *Research Issues in Data Engineering (RIDE)*, pp. 151–158. IEEE, 2002.

[129] C. Schnorr. Efficient Signature Generation for Smart Cards. *Journal of Cryptology*, **4**(1991), pp. 161–174.

[130] A. Shamir. How to Share a Secret. *Communications of the ACM*, **22**(1979), pp. 612–613.

[131] A. Spyropoulos and C. Raghavendra. Energy Efficient Communications in Ad Hoc Networks Using Directional Antennas. In *Proceedings of IEEE INFOCOM '02*, pp. 220–228. IEEE, 2002.

[132] V. Srinivasan, P. Nuggehalli, C.-F. Chiasserini, and R. Rao. Cooperation in Wireless Ad Hoc Networks. In *Proceedings of IEEE INFOCOM 2003*, pp. 808–817. IEEE, 2003.

[133] M. Stemm and R. H. Katz. Vertical Handoffs in Wireless Overlay Networks. *Mobile Networks and Applications*, **3**(1998), pp. 335–350.

[134] D. Stinson and T. Trung. Some New Results on Key Distribution Patterns and Broadcast Encryption. *Designs, Codes and Cryptography*, **14**(1998), pp. 261–279.

[135] J. Strunk, G. Goodson, M. Scheinholtz, C. Soules, and G. Ganger. Self-Securing Storage: Protecting Data in Compromised Systems. In *Proceedings of the 4th Symposium on Operating Systems Design and Implementation*, pp. 165–180. ACM, 2000.

[136] A. Stubblefield and D. S. Wallach. Dagster: Censorship-Resistant Publishing without Replication. Technical Report TR01-380, Rice University, 2001.

[137] C.-K. Toh. *Ad Hoc Mobile Wireless Networks: Protocols and Systems*. Prentice Hall, 2001.

[138] J. Vaidya and C. Clifton. Privacy Preserving Association Rule Mining in Vertically Partitioned Data. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 639–644. ACM, 2002.

[139] M. Waldman, A. Rubin, and L. Cranor. Publius: A Robust, Tamper-Evident, Censorship-Resistant, Web Publishing System. In *Proceedings of 9th USENIX Security Symposium*, pp. 59–72. USENIX, 2000.

[140] J. E. Wieselthier, G. Nguyen, and A. Ephremides. Energy-Limited Wireless Networking with Directional Antennas: The Case of Session-Based Multicasting. In *Proceedings of IEEE INFOCOM '02*, pp. 190–199. IEEE, 2002.

[141] D. Wikström. How to Break, Fix, and Optimize "Optimistic Mix for Exit-Polls". Technical Report T2002-24, Swedish Institute of Computer Science, 2002.

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

[143] S. Zhong. Privacy-Preserving Algorithms for Distributed Mining of Frequent Itemsets. Technical Report Yale DCS/TR1255, Yale University, 2003.

[144] S. Zhong, J. Chen, and Y. R. Yang. Sprite: a Simple, Cheat-Proof, Credit-based System for Mobile Ad Hoc Networks. In *Proceedings of IEEE INFOCOM 2003*, pp. 1987–1997. IEEE, 2003.

[145] S. Zhong and Y. R. Yang. Verifiable Distributed Oblivious Transfer and Mobile Agent Security. In *Proceedings of the DIALM-POMC Joint Workshop on Foundations of Mobile Computing*, pp. 12–21. ACM, 2003.

[146] L. Zhou and Z. J. Haas. Securing Ad Hoc Networks. *IEEE Network Magazine*, **13**(1999), pp. 24–30.