A Mobile Agent-based Privacy Protection Mechanism in Solving Multi-party Computation Problems

Md. Nurul Huda

DOCTOR OF
PHILOSOPHY

Department of Informatics,
School of Multidisciplinary Sciences,
The Graduate University for Advanced Studies (SOKENDAI)

2007 (School Year)

March 2007

A dissertation submitted to

Department of Informatics,

School of Multidisciplinary Science,

The Graduate University for Advanced Studies (SOKENDAI)

in partial fulfillment of the requirement for

the degree of Doctor of Philosophy

Advisory Committee:

Prof. Shigeki Yamada	National Institute of Informatics, SOKENDAI
Assoc. Prof. Yusheng Ji	National Institute of Informatics, SOKENDAI
Prof. Masatoshi Kawarasaki	Tsukuba University
Assoc. Prof. Hitoshi Okada	National Institute of Informatics, SOKENDAI
Prof. Noboru Sonehara	National Institute of Informatics, SOKENDAI

(Alphabetical order of list name except chair)

Abstract

A multi-party computation (MPC) allows n parties to compute an agreed-upon function of their inputs and every party learns the correct function output. To solve a multi-party computation problem (MPCP), the participants may need to share their private data (inputs) between one another, resulting in data privacy loss. The key research issue that has been addressed in this thesis is - how to solve multi-party computation problems without disclosing anyone's private data to others.

Firstly, by studying and analyzing the traditional computational models, we have devised a privacy loss model for multi-party computation problems and proposed a novel metric, called the Min privacy metric, for quantitatively measuring the amount of data privacy loss in solving the MPCPs. Then, we have presented a mobile agent-based scheduling algorithm that applies pseudonymization technique to reduce data privacy loss. Finally, we have proposed the security system design, including security policies and security architecture, of an agent server platform for enhancing data privacy protection while solving the MPCPs.

The privacy loss model has identified three factors affecting the amount of privacy loss in solving the MPCPs: (1) the fraction of private data which is shared with others, (2) the probability of associating the shared private data with the data subject, and (3) the probability of disclosing the shared private data to unauthorized parties. Privacy loss can be reduced by any mechanisms which reduces the values of any of the three factors. The proposed Min privacy metric accounts for the number of participants that lose their private data and the amount of private data disclosed to unauthorized parties, regardless of how many parties they are revealed to.

Existing scheduling algorithms aim for a global objective function. As a result, they incur performance penalties in computational complexity and data privacy. This thesis describes a mobile agent-based scheduling scheme called Efficient and Privacy-aware Meeting Scheduling (EPMS), which results in a tradeoff among complexity,

privacy, and global utility for scheduling multiple events concurrently. We have introduced multiple criteria for evaluating privacy in the meeting scheduling problem. A common computational space has been utilized in EPMS for reducing the complexity and pseudonymization technique has been applied to reduce the privacy loss in the scheduling problem. The analytical results show that EPMS has a polynomial time computational complexity. In addition, simulation results show that the obtained global utility for scheduling multiple meetings with EPMS is close to the optimal level and the resulting privacy loss is less than for those in existing algorithms.

Cryptography-based algorithms for MPCPs are either too complex to be used practically or applicable only to the specific applications for which they have been developed. In addition, traditional (non-cryptography-based) algorithms do not provide good privacy protection for MPCPs. We have proposed a novel privacy protection mechanism in which MPCPs are solved by mobile agents using traditional algorithms at an agent server platform, called isolated Closed-door One-way Platform (iCOP). The participating mobile agents are trapped into iCOP where they are allowed to share their private information to solve the problem using traditional algorithms. However, they are protected from disclosing the shared private information to the outside world. The enforcement of the security policies protects the participating agents from sending anything other than the computational result to the users. The security and privacy analysis illustrates that the proposed mechanism provides very good privacy protection if the participants solve the problem with distributed algorithms and can provide complete privacy protection if the participants exchange inputs within the iCOP and each of them solve the problem with centralized algorithms. Finally, experimental evaluation shows that the proposed agent platform security system significantly enhances privacy protection while solving many MPCPs with traditional algorithms.

Keywords: Multi-party Computation, Privacy, Mobile Agent, EPMS, iCOP, Covert Channel, Steganography, Distributed Computing.

Student Number: 20041707

Acknowledgements

I wish to express deepest gratitude to my supervisor Prof. Shigeki Yamada and Assistant Prof. Eiji Kamioka for their patient guidance with candid comments, creative suggestions, and numerous discussions on several of my research paper manuscripts and on the thesis as a whole, without which this research would not have been feasible. My sincere thanks also goes to my thesis advisory committee members, Prof. Masatoshi Kawarasaki, Prof. Noboru Sonehara, Associate Prof. Yusheng Ji and Associate Prof. Hitoshi Okada for their valuable suggestions and comments to improve the quality of my research, as well as this thesis.

I would like to thank my colleagues, Hasanuzzaman, Thepparit Banditwattanawong, and Ved Prasad Kafle for their friendly assistance and the authors of some important research papers for their help through e-mails in understanding their papers.

I would like to thank the staff in the Graduate Students Section, National Institute of Informatics (NII) for their assistance and suggestions that made my NII life comfortable. I am grateful to NII and Japan Student Services Organization (JASSO) for their generous scholarship support.

Last but not least, I am grateful to my beloved wife, Bonhomie, and son, Adrian, for enduring loneliness while I could not stay with them for my research work. I thank my parents, brothers and sisters for their encouragement which motivated me to work and complete my education up to this Ph.D. degree.

March 25, 2007

Md. Nurul Huda

Table of Contents

List of Figures

List of Tables

Acronyms

ADOPT Asynchronous Distributed Optimization

AS Authentication Server

CBR Cycle-based Runtime

CS Computation Server

DCOP Distributed Constraint Optimization

DisCSP Distributed Constraint Satisfaction Problem

EPMS Efficient and Privacy-aware Meeting Scheduling

GU Global Utility

iCOP isolated Closed-door One-way Platform

JASSO Japan Student Services Organization

MPC Multi-party Computation

MPCP Multi-party Computation Problem

MS Meeting Scheduling

NII National Institute of Informatics

optAPO optimal Asynchronous Partial Overlay

SMC Secure Multi-party Computation

TTP Trusted Third Party

UF Utility Factor

VPS Valuation of Possible States

Thesis Overview

The proliferation of the Internet has opened the opportunities for automated cooperative computation, where people are cooperating with each other to conduct computation tasks based on the inputs they each supplies. These computations could occur between trusted parties, between partially trusted parties, or even between untrusted parties. For example, two competing financial organizations might jointly invest in a project that must satisfy both organizations' private and valuable constraints, two parties want to carry out statistical analysis with their private databases to get mutual benefit in a collaborative project, and so on. Usually, to conduct these computations, one must know inputs from all the parties; however if nobody can be trusted enough to know all the inputs, privacy will become a primary concern.

The Research Issue

Two or more parties want to conduct a computation based on their private inputs, but neither party is willing to disclose its own inputs to anybody else. The problem of how to conduct such a computation while preserving the privacy of the inputs is referred to as Secure Multi-party Computation (SMC) problem in the literature. Generally speaking, a secure multi-party computation problem deals with computing a joint function on any input, in a distributed network where each participant holds one of the inputs, ensuring independence of the inputs, correctness of the computation, and that no more information is revealed to a participant in the computation than what can be inferred from that participant's input and output. We address the data privacy issue of the participants of a multi-party computation and want to develop a mechanism for solving multi-party computation problems without disclosing any participant's private inputs to the others.

Goal of this Research

The primary goal of this research work is find a mechanism for protecting private (input) data while solving multi-party computation problems without having too much complexity. The mechanism should not be application-specific so that it can be applied for many applications. In addition, it should not incur very high performance penalties in other important considerable factors (e.g., computational time), which might make it impractical.

Existing cryptography-based generalized solutions are too complex to be used practically. Even though cryptography-based problem-specific solutions are more efficient than the generalized solutions, the applicability of each of these cryptography-based algorithms is limited to only the specific problem for which it has been developed. We need an efficient generalized solution which can be used for many applications. Besides, very efficient algorithms exist for many multi-party computation problems in which privacy has not been taken into consideration. We want to utilize those algorithms for solving the MPCPs. Thus, we want to develop a non-cryptography based privacy protection mechanism for multi-party computation problems so that existing traditional (non-cryptography based) algorithms can be used.

The Overall Approach

We have achieved low privacy loss in a mobile agent based scheduling algorithm by using a common computational space. In the proposed scheme, the participating mobile agents, with their respective user's personal information, migrate into a common agent platform, controlled by an access control mechanism. They encode their hard constraints and soft constraints into numerical values and build a Utility Factor (UF) vector at a common computational space at the agent platform. They gradually update and build the utility factor vector in random sequence without revealing the current updating participant's identity to the others. The utility factor vector of an event reflects all feasible solutions for that event and at the same time group preference (i.e., utility) for those feasible solutions. An agent then finds out the most

preferred time slot for the event from all feasible solutions via simple search. In case of scheduling multiple events, an agent calculates the scheduling priorities of its events based on the maximum utility found in the UF vectors and then schedules them based on their priorities. The use of a common computational space enables us to utilize pseudonymization technique to preserve data privacy and also makes our algorithm very simple having low computational complexity.

We have presented a mobile agent-based privacy protection mechanism in which the participating mobile agents take users' private data that are required for the desired computation and then, along with that data, migrate into an agent server (provided by a third-party service provider), called isolated Closed-door One-way Platform (iCOP), into which the mobile agents are trapped. To perform the desired computation, the agents exchange messages locally and solve the problem by sharing their private data within the server. The privacy manager of the proposed server architecture restricts the participating agents from disclosing the acquired private data of other agents to the outside world. It also restricts the participating agents from leaving the platform with the shared private data. A stationary trusted agent, called the service agent, sends the computational result to the users after the required verification that the computational result does not contain any hidden data in it. Finally, in order to destroy the shared data, the participating agents along with their acquired data, are terminated (or disposed) at the agent server.

We investigate possible data disclosure channels at iCOP and describe how whose channels are handled in our proposed mechanism. We also evaluate the proposed privacy protection mechanism by comparing the privacy loss and computational cost in it with those in the traditional systems for the same algorithm.

Contributions of the Research

Firstly, after studying the traditional computation models, we have devised a privacy loss model for multi-party computation and identified three factors for privacy loss that give a general guideline for mechanisms of reducing privacy loss. We have also

proposed a novel privacy metric, called the Min privacy metric, for measuring privacy loss quantitatively. Then, we have developed a mobile agent-based scheduling scheme called Efficient and Privacy-aware Meeting Scheduling (EPMS), which has low computational complexity, results in a global utility close to the optimal level necessary, and achieves better privacy protection by utilizing a common computational space. We have also introduced multiple criteria for evaluating privacy in the scheduling algorithm.

We have proposed a mobile agent-based privacy protection mechanism for multi-party computation problems, the security policies for the participants, and the security architecture of an agent server platform for enforcing those security policies. We prove through intuitive analysis that the enforcement of our proposed security policies for the agent server platform can provide a very good privacy protection in solving many MPCPs with traditional (i.e. non-cryptography-based) algorithms. Also, we have shown a mechanism for achieving complete privacy protection.

Organization of the Thesis

The rest of the thesis is organized as follows. In Chapter 1 we describe the multi-party computation problem with examples and discuss its privacy issues. Chapter 2 illustrates the related previous works and their limitations. In Chapter 3 we devise the privacy loss model showing the factors that affect privacy loss and propose a privacy measurement metric. Chapter 4 formalizes the meeting scheduling problem, presents the EPMS algorithm, analyzes its complexity, privacy loss, global utility, and computational cost. Chapter 5 describes the proposed mobile agent-based privacy protection mechanism in details, carries out its security and privacy analysis, shows its effectiveness by comparing the privacy loss for it with those in the traditional mechanisms. Chapter 6 discusses about some of the applications of the proposed EPMS algorithm and the proposed mobile agent-based privacy protection mechanism. Finally, Chapter 7 concludes the thesis with a description of the summary of the main results and future works.

Chapter 1

Introduction

Privacy has many different meanings in different aspects. One of the earliest legal definition of privacy was given by Warren and Brandeis in [62], who identified privacy as "the right to be left alone". This definition is very general and includes various issues concerned with physical intrusion that we will not address. Westin in [63] defines privacy as "the claim of individuals, groups and institutions to determine for themselves, when, how and to what extent information about them is communicated to others". This definition is well accepted by consumer rights groups and in concordance with the information related parts of the definitions given by others [45]. Rosenberg in [50], presents three aspects of privacy: (i) territorial privacy, which is concerned with the physical area of a person (e.g., his home); (ii) privacy of the person, which is concerned with direct physical access to a person (e.g., a physical search); (iii) informational privacy, which is concerned with information about a person (e.g., the processing of personal data). In the context of this thesis, by the term "privacy" we refer only to the aspect of informational privacy (or data privacy). We believe that this restriction causes no problems for our application domain due to the inherent lack of physical interaction in the use of communication and information systems.

Privacy and security are forever entwined, but they are not the same. Generally, data security is the means of ensuring that data are kept safe from corruption and that access to them is suitably controlled. Data security helps to ensure privacy. Without security, there can be no privacy. Privacy, in addition, deals with authorized use of private data by correct authority and protection from unauthorized parties during their use.

In the computation involving information sharing the principle problem of information disclosure is directly related to privacy. Once some information is shared with another principal, the original holder of the information loses all control over it. The receiver can use it without any constraints, duplicate it, or disclose it to an

unauthorized party. This is a fundamental problem, for which no efficient generalized technical solution exists. A solution is said to be privacy-preserving or secure if no participant can learn more from the description of the joint function and the result of the calculation than what he/she can learn from his/her own entry and the output.

1.1 Multi-party Computation Problems

In order to solve multi-party computation problems (MPCP), personal information of some of the parties is required to be shared with other parties. Thus, the problem of information disclosure is a primary concern in MPCP. In a multi-party computation, we have a given number of participants or parties $p_1, p_2, ..., p_n$, each having a private data, $d_1, d_2, ..., d_n$ respectively. The participants want to compute the value of a joint function f on n variables (Fig. 1.1).

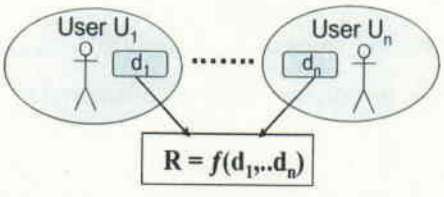


Figure 1.1: Multi-party computation problem

For example, Alice has m private numbers $(a_1, a_2, ..., a_m)$ and Bob has another m private numbers $(b_1, b_2, ..., b_m)$. Alice and Bob want to know whether $a_i > b_i$ for all $i = 1, ...m$. This problem is known as the vector dominance problem [10]. There are many applications to the vector dominance problem. For example, in business-to-business bidding, a manufacturer may want to deal with a single supplier that can simultaneously satisfy all of the m requirements (either because there is some coordination required in the production of m item types, or simply to avoid the bureaucratic overhead of having to deal with multiple suppliers).

For another example, consider the following situation: after a costly market research, company A decided that expanding its market share in some region will be very beneficial. However, A is aware that another competing company B is also planing to expand its market share in some region. Strategically, A and B do not want to

compete against each other in the same region, so both the companies want to know whether their regions overlap.

A large group of multi-party computation problems are classified as the distributed constraint satisfaction problem (DisCSP) [53, 59, 65] and the distributed constraint optimization problem (DCOP) [44, 46]. They are solved with the assistance of software agents. A DisCSP consists of a number of variables $x_1, x_2, ..., x_n$, whose values are taken from finite, discrete domains D_1, D_2, ..., D_n, and a set of constraints on their values. There are inter-agent constraints and inra-agent constraints. Solving a DisCSP is equivalent to finding an assignment of values to all variables such that all constraints are satisfied [65]. In a distributed algorithm, the participants exchange information related to their value assignment to the variables to check if the constraints are satisfied. Additionally in DCOPs, a global objective function is optimized. The exchange of information related to the value assignment to the variables is analogous to negotiation with each other and (re)assigning values to all variables is like reaching an agreement upon the value assignment to the variables that satisfies the constraints.

As an example of DisCSP, consider that a number of employees of an organization want to schedule a new meeting in a predetermined time domain. For this job, they need feedback from their personal calendar information to avoid conflict with other personal schedules and look for available time slots in the specified time domain and personal preferences for better user satisfaction. They want to find an agreed upon meeting date/time slot that satisfies all the constraints (e.g. does not conflict with other personal schedules). In a DCOP, the solution also maximizes or minimizes some global objective function (e.g. most preferred, least cost) of the group.

Since, the multi-party computation problems deal with common problems among a group of participants, the computational result of many multi-party computation problems can be expressed with a common value for all of the participants. For example, in the vector dominance problem, the computational result for both the participants can be expressed with a common value indicating whether vector A dominates vector B. Similarly, in the meeting scheduling problem, all of the participants need to find a common time slot for the meeting. In this thesis, we take into consideration a type of multi-party problems in which the same computational result can be disclosed

to all of the participants without causing privacy loss. The multi-party problems with the above assumption are not trivial but typical, and many multi-party applications are covered with the above assumptions.

1.2 Privacy Issue in Multi-party Computation Problems

Generally speaking, the participants of a multi-party computation problem get mutual benefit and hence are assumed to be collaborative. Even though they want to solve the problem, in many contexts, they may want to keep their inputs secret from one another. The encrypted form of the input data cannot be used in traditional algorithms for various types of computation. So, for usability, data need to be kept in normal (non-encrypted) form. Sharing private data (in normal form) with others causes privacy loss. The privacy challenge in multi-party computation problems is to share data among participants for the intended computation, while protecting future use or disclosure of the data to unauthorized parties.

For instance, in the market share expanding scenario mentioned Section 1.1, both the companies want to know whether their regions overlap. However, they do not want to give away their location information (not only would disclosure of this information cost both companies a lot of money, it can also cause significant damage to the company if it is disclosed to other parties, e.g. another bigger competitor could then immediately occupy the market there before A or B even starts). Therefore, they need a way to solve the problem while maintaining the privacy of their location information.

In the vector dominance problem, Alice and Bob want to determine if Alice's bidding vector A dominates Bob's price vector B. However, both Alice and Bob do not want the other party know anything about her/his vector including the elements of the vectors or the relationship between any element of vector A with any element of vector B or how many of A's elements are bigger than the corresponding elements in B.

In the meeting scheduling problem, the personal calendar information and personal preferences of individual participants are needed for making an efficient schedule. But those information are considered private information. Even though the participants

want to make the schedule, they may not want to disclose those private information to others.

The privacy problem is easy to solve if you assume the existence of a trusted third party (TTP), which can be trusted by all of the participants and the participants are willing to disclose all of the inputs to the trusted third party. The TTP computes the function and distributes the result to everyone (Fig. 1.2). However, the problem is very challenging if you assume that there is no TTP available to which the participants may want to disclose their private inputs.

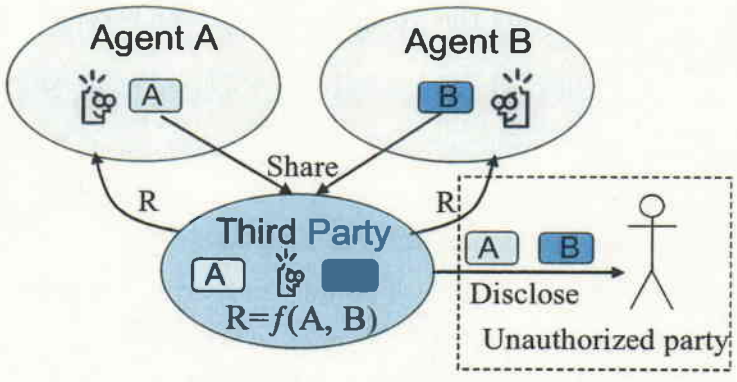


Figure 1.2: Third-party approach for solving the MPCP.

Thus, some privacy-aware solutions of multi-party computation problems try to solve the problem by not disclosing the private data as much as possible. For example, distributed algorithms solve the problem by not disclosing all of the private data to others. Some of the private data are disclosed to some other participants. So, the resulting privacy loss is comparatively low.

Cryptography based algorithms try to solve the problem by keeping the private data in an encrypted form. They can provide complete privacy protection. However, their usability is very limited.

Existing cryptography-based generalized solutions are too complex to be used practically. Researchers have developed a few problem-specific cryptography-based algorithms for solving specific MPCPs [11][51][64] based on the specific characteristics of the related specific problem. Even though the problem-specific solutions are more efficient than the generalized solution, the applicability of each of these cryptography-based algorithms is limited to only the specific problem for which it

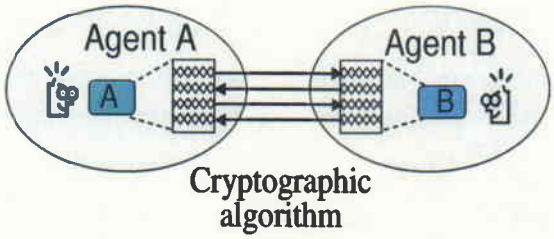


Figure 1.3: Cryptographic approach for solving the MPCP.

has been developed [51]. Thus, we can conclude that an efficient generalized solution for protecting privacy in solving the MPCPs has not been evolved yet.

Chapter 2

Background and Related Works

The problems of conducting multi-party computation while preserving privacy of the inputs is referred to as Secure Multi-party Computation Problem (SMC) in the literature [64]. This problems was initially suggested by Andrew C. Yao in 1982 by introducing the millionaire problem: Alice and Bob are two millionaires who want to find out which is richer without revealing the precise amount of their wealth. Yao proposed a solution allowing Alice and Bob to satisfy their curiosity while respecting the constraints.

To solve various secure multi-party computation problems, a common strategy is to assume the trustworthiness of the service providers, or to assume the existence of a trusted third party, to whom the inputs can be disclosed. However, this solution may not be always acceptable. Therefore, the solutions that can support multi-party computations while protecting the participants' privacy are of growing importance.

In theory, the general secure multi-party computation problem is solvable [64, 21, 22] but, as Goldreich points out in [21], using the solutions derived by these general results for special cases of multi-party computation can be impractical; special solutions should be developed for special cases for efficiency reasons.

Since the introduction of secure multi-party computation problem, many researchers have provided different solutions. We can classify them into two groups (1) Cryptographic approaches and (2) non-cryptographic approaches.

2.1 Cryptographic Approaches

Cryptographic approaches for solving multi-party computation problems are based on many cryptographic tools including zero-knowledge proof [52], oblivious transfer [48], 1-out-of-n oblivious transfer [14, 6], oblivious evaluation of polynomials [47], secret sharing [54], threshold cryptography [17, 9], and Yao's millionaire protocol [64, 7].

Goldwasser [22] and Goldreich [21] provide comprehensive overview articles on secure multi-party computation.

Goldreich, Micali, and Wigderson [21] and many others have extended Yao's work on SMC. These works all use a similar methodology: the computation problem is first represented as a combinatorial circuit, and then the parties run a short protocol for every gate in the circuit. While this approach is appealing in its generality and simplicity, the protocols it generated depend on the size of the circuit. This size depends on the size of the input domain, and on the complexity of expressing such a computation. The obvious drawback of the general solutions is efficiency: All proposed protocols for secure multi-party computation suffer from high communication complexities. This follows from their approaches which involve extensive use of secret sharing and agreement protocols. For example, in [29] it was shown that any circuit with S gates can be computed unconditionally secure and t-robust for $t < n/3$ with communication complexity $O(Sn^2k+n^2BC)$, where k is the size of the elements of the field secret-sharing is done over and BC is the communication complexity of broadcasting a k-bit value. Recently, similar communication complexities were achieved for $t < n/2$, once with cryptographic security ($O(Sn^2k + nBC)$, where k is the security parameter, [30]), and once with information-theoretic security ($O(Sn^2k + n^3BC)$), [60]. Also, privacy protection has been achieved in distributed constraint satisfaction problems by using cryptographic techniques [66]. However, this technique also incurs large overhead and requires to use multiple external servers which may not always be justifiable for its benefit.

The privacy-preserving data mining problem is another specific secure multi-party computation problem that has been discussed in the literature. In Lindell's paper [40], the problem is defined as: two parties, each having a private database, want to jointly conduct a data mining operation on the union of their two databases. How could these two parties accomplish this without disclosing their database to the other party, or any third party. Apart from the above problems, secure multi-party computation problems exist in many other computation domains as well [1, 10, 12].

2.2 Non-cryptographic Approaches

The existing non-cryptography based techniques for solving the multi-party computation problems are very simple but do not provide complete privacy protection. The traditional centralized approach (i.e., the trusted third party approach) requires to disclose all of the inputs to a central entity causing high privacy loss to the central entity.

Reducing privacy loss is one of the key motivating factors in many distributed algorithms. The participants of a distributed problem need to share their private valuation of certain variables to resolve the conflicts or contradictions among them in distributed constraint satisfaction problems and also to optimize the solution in distributed constraint optimization problems. Existing distributed algorithms themselves cannot offer complete privacy protection [25][41][66]). However, a distributed algorithm offers better privacy protection than a centralized algorithm. Different distributed algorithms results in different level of privacy loss. Also, depending upon the metrics used for privacy loss measurement, the relative privacy loss in an algorithm may vary with respect to other algorithms [25][41].

The main technique used for reducing privacy loss in most distributed algorithms is to reduce the amount of shared information with the others. However, all these algorithms have some inherent privacy loss. To resolve the conflicts among the variables or to optimize the solution, the participants cannot avoid sharing their private inputs completely.

One method to reduce privacy loss is to remove or hide the relationship of the personal data with the identifiable individual [2][58]. This is commonly referred to as anonymization. In a multi-party computation and for a known number of participants, the relationship of the data with an individual is bounded by the number of individual (n) i.e., some private information can be associated with an individual with the probability of no less than $1/n$. This approach is more suitable in privacy preserving data mining where there are a large number of individuals related to the data set.

In some situations, only part of the data set needs to be kept confidential. For example, when two retail stores want to conduct a joint computation on their joint

data, they are only concerned about their customers' names, not about each single transaction. In these cases, the problems could be solved using pseudonyms techniques [4, 5] in which the identity of individuals are replaced with some artificial identifiers.

Chapter 3

Privacy Loss Model

Software agent technology plays a very important role in making autonomous system. Autonomy is more essential specially in complex multi-party computation problems, such as the distributed constraint satisfaction (DisCSP) problem [65][66] and distributed constraint optimization (DCOP) problem [44][46], that otherwise is a tedious and time-consuming job for human. Multi-party computation problems (including DisCSPs and DCOPs) are generally dealt with software agents, which can represent the users, and can negotiate and take decisions on behalf of the users. For example, scheduling meetings among a number of participants, in the presence of communication delays and other meetings being scheduled concurrently, is a time-consuming, iterative, and tedious job for humans and may lead to inefficient solutions. Automating meeting scheduling with software agents can save time and effort, and may produce more efficient schedules.

3.1 Agent-based Computing

Traditional agent based systems are multi-agent systems in which the agents reside in a distributed network. The agents communicate with one another through remote messages. Thus, when an agent shares its (user's) private data with another agent, the sender agent retains no control over the given data. Consider the following scenario: two participants A and B want to compute a joint function $f(A, B)$, which involves their private inputs 'A' and 'B' respectively. When agent A shares its private data 'A' with agent B, agent B can disclose the shared data 'A' to its user B (Fig. 3.1). Similarly, when agent B shares its private data 'B' with agent A, agent A can disclose the shared data 'B' to its user A (Fig. 3.1).

Inter-agent communication takes place point-to-point and the data receiver agent

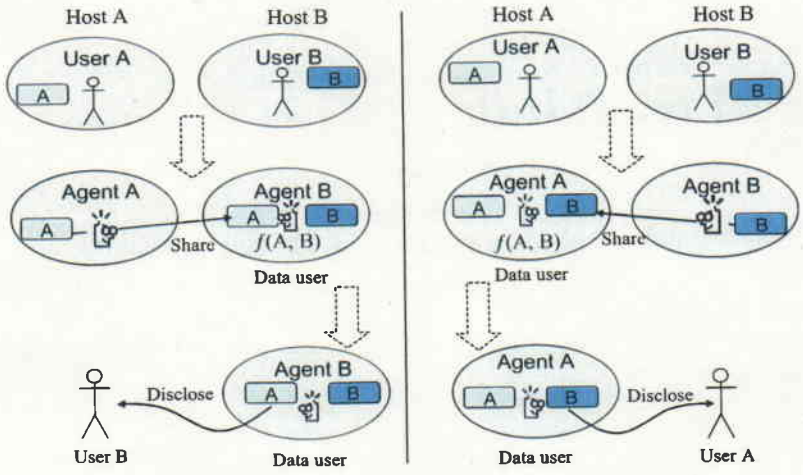


Figure 3.1: Data sharing in traditional multi-agent systems.

can easily identify the data sender agent. The communicated private data are inherently associated with the sender agent and its associated user. So, the data receiving agent, as well as its associated user, can always identify the data subject.

Cryptographic solutions to the multi-party computation problems usually require a large number of message communication [29, 66], which create a large overhead on the network and require a large computation time. In a mobile agent based system, the participating agents can migrate into a common agent platform and communicate locally from within the agent platform. For example, in Fig. 3.2, mobile agents in

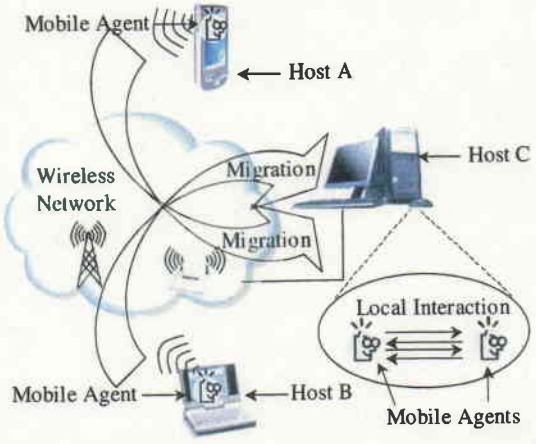


Figure 3.2: Mobile agent based system for multi-party computation problems.

hosts A and B may migrate into host C to communicate locally and carry out their

computation in host C. This can reduce the network load and the communication latency (i.e., computational time). Mobile agent paradigm offers a number of potential advantages over traditional multi-agent system, especially in wireless network environments [39]. Thus, it is preferable to use mobile agent technology for those problems that involve large number of messages.

3.2 Devised Privacy Loss Model

The privacy loss model of multi-party computation consists of three parties: data subject or owner, data user, and unauthorized party. The data owner or subject is the entity about whom the data carry some (private) information. The data user is the entity, who utilizes the shared data for performing the desired computation. The unauthorized party is the entity to whom the private data should not be disclosed. When part of the private data is shared with the data user, that shared data goes into the data user's system from where it may be disclosed to unauthorized parties (Fig. 3.3). However, the acquisition of private data by an unauthorized third party does not necessarily mean privacy loss. If the unauthorized party cannot map or associate the acquired data with the specific individual we do not say it is privacy loss.

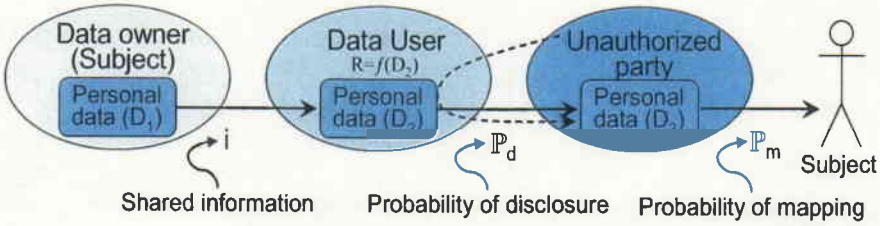


Figure 3.3: Privacy loss model.

Therefore, we can define privacy loss from the data user's system as,

$$\mathcal{P} = \mathbb{P}_d \times \mathbb{P}_m \times i \tag{3.1}$$

where, $\mathbb{P}_d$ is the probability of disclosure of the shared data from the data user's system, $\mathbb{P}_m$ is the probability of associating or mapping the acquired data with the identifiable individual by the unauthorized party, and i is the size of the shared data, which is shared with the data user or visible to the data user.

Let the ideal privacy level of a user be a one and the worst case privacy level be a zero. With this normalization, the privacy loss becomes,

$$\mathcal{P} = \alpha \times \mathbb{P}_d \times \mathbb{P}_m \times i \tag{3.2}$$

where, α is an appropriate scaling factor whose value depends upon the size of the private data required for the computation. With $\mathcal{P}$ privacy loss, the privacy level of the user becomes,

$$\tilde{\mathcal{P}} = 1 - \mathcal{P} = 1 - \alpha \times \mathbb{P}_d \times \mathbb{P}_m \times i. \tag{3.3}$$

From Eq. (3.2) it can be said that privacy loss takes place if, (a) the shared data is disclosed by the data user to any unauthorized entity (i.e., $\mathbb{P}_d > 0$) and (b) the unauthorized data receiver can map or associate the private data with the specific individual (i.e., $\mathbb{P}_m > 0$). In conventional multi-agent based distributed systems, the agents are the data users and they communicate with one another from their own hosts administered by the agent owner/user. This sharing of private data from one agent to another is considered a privacy loss in conventional multi-agent based systems, because (a) the data receiving agent, which is administered by its owner or user, is free to disclose the acquired private data to its user (i.e. $\mathbb{P}_d = 1$) and (b) the receiving agent can map or associate the received private data, though point-to-point communication, with the data sender agent or data owner (i.e. $\mathbb{P}_m = 1$).

Different private data will have different valuation depending upon their sensitivity. Also, the same private data may be very important for one user, while they may not be of that much important to another user. Thus, the amount of privacy loss may be estimated differently by different users depending upon their private valuation of the private data. So, we introduce a weight factor ω in the privacy loss measurement, which can take a value in the [0..1] span depending upon the valuation of the private data of the data subject. Thus, the privacy loss of a user can be expressed as,

$$\mathcal{P} = \omega \times \alpha \times \mathbb{P}_d \times \mathbb{P}_m \times i. \tag{3.4}$$

It is assumed that none of the participants of a multi-party computation wants to disclose her private inputs to the others. In other worlds, the value of ω has been

considered to be greater than 0 for all of the participants. For simplicity, we consider the value of ω to be equal to 1 for all of the participants.

Privacy loss can be reduced by mechanisms that reduce the value of $\mathbb{P}_d$ and/or $\mathbb{P}_m$ and/or i. Among different approaches to reduce the privacy loss, anonymization and pseudonymization techniques reduce the value of $\mathbb{P}_m$, and cryptography-based algorithms and some distributed algorithms reduces the amount of information i disclosed to the data user.

3.3 Evaluation Metric

In this section, we describe the metrics used in this thesis to measure parameters (e.g. privacy loss and computational time) for performance evaluation of different approaches.

3.3.1 Privacy Loss Measurement Metric

There is a need for a quantitative framework that would allow us to express metrics for measuring privacy loss. The authors of paper [42] have presented a framework, called the Valuation of Possible States (VPS), for qualitatively analyzing the privacy loss among collaborative agents. According to the VPS framework, the private information of an agent can be modeled as a state among a set of possible states. Let the private information of agent A_i be modeled as a state $s_i \in S_i$, where S_i is a set of possible states that agent A_i may occupy. If there are n participants in a computation, then there would be $(n-1)$ observing agents for each observed agent. If an agent has d number of states, each represented by a data unit, then that agent has a total of $(d*(n-1))$ states to all observing agents.

VPS privacy metric: Privacy loss measurement counts the number of states revealed to the observing agents and the the number of observing agents to whom the states are revealed. If all of the $(d*(n-1))$ states are revealed to the $(n-1)$ observing agents, and d states to each of them, the privacy loss is 'one' (Fig. 3.4). If no state of an agent is revealed to any observing agent, the privacy loss is 'zero'. Let us call the privacy metric in the VPS framework the "VPS privacy metric".

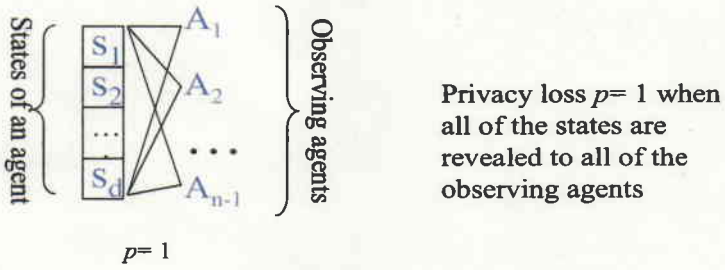


Figure 3.4: The VPS privacy metric.

Let $\mathbb{P}_i^m(s_i(T_j))$ be the probability of characterizing A_i into a state $s_i \in S_i$ for the data unit T_j by any other observing agent A_m. Then, according to the VPS framework, the privacy metric can be expressed as,

$$\mathbb{V}_i(\mathbb{P}_i(S_i)) = \sum_{m \neq i} \sum_{s_i \in S_i} \sum_{j=1}^{d} I_{\{\mathbb{P}_i^m(s_i(T_j))>0\}} \tag{3.5}$$

where, $I_{\{.\}}$ is an indicator function. The privacy metric $\mathbb{V}_i$ counts the number of states of an agent A_i revealed to all other observing agents.

In a general multi-agent environment, $\mathbb{P}_d = 1$, since the receiving agent is free to disclose the acquired private data to its user and $\mathbb{P}_m = 1$, since the receiving agent can always know the sender (i.e., owner) of the shared private data. So, in a [0..1] privacy span, the privacy loss $\mathcal{P}_i$ of agent A_i becomes,

$$\begin{aligned} \mathcal{P}_i &= \alpha \times \mathbb{V}_i(\mathbb{P}_i(S_i)) \\ &= \alpha \times \sum_{m \neq i} \sum_{s_i \in S_i} \sum_{j=1}^{d} I_{\{\mathbb{P}_i^m(s_i(T_j))>0\}} \end{aligned} \tag{3.6}$$

where, α is an appropriate scaling factor whose value depends upon the number of subject states. With $\mathcal{P}_i$ privacy loss, the privacy level of agent A_i becomes,

$$\tilde{\mathcal{P}}_i = 1 - \alpha \times \sum_{m \neq i} \sum_{s_i \in S_i} \sum_{j=1}^{d} I_{\{\mathbb{P}_i^m(s_i(T_j))>0\}}. \tag{3.7}$$

Proposed Min privacy metric:

The VPS metric takes into consideration that the private information could be disclosed to only the participants of the MPCP. However, it is reasonable to consider

that the number of unauthorized parties might be unknown. Also, the leaked information can be propagated to unknown number of parties. Thus, in many cases it is more appropriate to measure the privacy loss based on the amount of information revealed to the others, regardless of how many of them the information is revealed to. We propose a privacy loss measurement metric that we call the "Min privacy metric", which is not a function of the number of unauthorized parties. In this metric, the amount of privacy loss remains the same if the same amount of information is revealed to even less number of observing agents. If all of the d states (private information) of agent A_i are revealed to any observing agent then its privacy loss is 'one', and the loss is 'zero' if no state is revealed to the observing agents. Figure 3.5 shows three scenarios in which different number of states of an agent are revealed to different number of observing agents. Even though the amount of privacy losses in the VPS privacy metric and in the Min privacy metric are the same for scenario 1, they are different for scenarios 2 and 3.

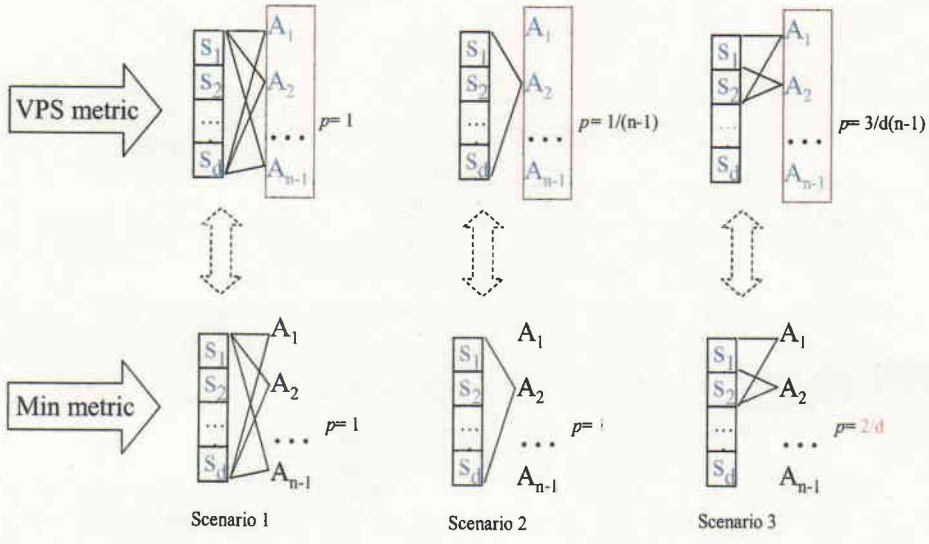


Figure 3.5: The Min privacy metric.

The Min privacy metric accounts for the number of participants that lose their information and the amount of information that they lose to any observing agents, regardless of how many observing agents it is revealed to. So, in the Min privacy

metric the privacy loss $\mathcal{P}_i$ of agent A_i is defined as,

$$\mathcal{P}_i = \alpha \times \sum_{s_i \in S_i} \sum_{j=1}^{d} I_{\{\mathbb{P}_i^{m \neq i}(s_i(T_j))>0\}} \tag{3.8}$$

where, α is an appropriate scaling factor whose value depends upon the total number of states of agent A_i that represent its private information.

The Min metric should be used to measure privacy loss when the number of unauthorized party is not definite and/or the leaked information may be propagated to other parties than the parties to whom the information is directly disclosed. Otherwise, if neither of the above conditions is met, the VPS metric can be used to measure privacy loss.

3.3.2 Runtime Measurement Metric

The runtime measurement for a centralized algorithm is straightforward and can be done by inspecting the actual time required for a single machine in executing the algorithm. However, the runtime for a distributed algorithm cannot be measured by inspecting the time required for single machine.

The acknowledged method of measuring synchronous execution of distributed algorithm is with discrete cycles [65]. However, the total cycle count does not tell us anything about the length of a cycle or the total runtime of the algorithm. Cycle-Based Runtime (CBR) [8] appears to be the most appropriate metric for measuring runtime of a distributed algorithm. Let L denote the time required in a cycle to deliver all messages sent in the previous cycle and $cc(x_i, k)$ be the number of constraint checks performed by agent x_i in cycle k. Then time for m cycles,

$$CBR(m) = L \times m + \sum_{k=0}^{m} max(cc(x_i, k)) \times t \tag{3.9}$$

Time required to transmit a message is usually greater than the time for a constraint check in most environments, so for simplicity we assume that a constraint check is the smallest atomic unit of time ($t = 1$), and assume L is given relative to t. By measuring the time for message delivery and the maximum number of constraint check in each cycle, we can measure relative Cycle Based Runtime of two algorithms.

Chapter 4

Reducing Privacy Loss using Common Computational Space

One method to reduce privacy loss is to remove or hide the relationship of the personal data with the identifiable individual [2][58]. This is commonly referred to as anonymization or pseudonymization. In this chapter we utilize a common computational space for making the pseudonymization effect to reduce privacy loss in the meeting scheduling problem.

4.1 The Meeting Scheduling Problem

The meeting scheduling (MS) problem is a collaborative multi-agent problem. Each participating agent, which represents a user, uses the respective user's personal calendar and preferences to find a meeting time when all of the participants are free and the group preference is maximized [15, 26, 61]. We use the terms "user" and "agent" interchangeably.

In the MS problem, the users' personal schedules and their preferences are considered private information [15][28][55][66]. In the centralized algorithm, all of the participants give their private information to a third party or one of the participating agents, who solves the problem on behalf of the participants and returns the result. This algorithm results in a high privacy loss of the participants to the central agent [25]. Centralization has the advantage of aggregating the preferences and constraints into a common computational space, which enables the algorithm to avoid large communication costs.

In distributed algorithms [15][26][44][46][55], the participating stationary agents communicate through remote messages and collaboratively try to find a solution to the problem. A traditional distributed MS algorithm can be viewed as a mechanism that uses proposals and replies to find solutions [15][16][66]. One of the agents proposes a

certain date/time to the others at which it is free. The other agents look into their own calendars and check whether they are free on the proposed slot and reply accordingly. The fact that an agent proposes a certain date/time to the other agents, the other agents can infer that the proposing agent does not have any personal schedule on that proposed date/time. Thus, they get some private calendar information of the proposing agent. Similarly, if an agent declines a certain proposal, the proposing agent can infer that the rejecting agent already has a schedule in that slot or some overlapping slot. The proposals and replies carry personal information, and they are inherently associated with the identifiable sender agents. Thus, they reveal a user's personal information to the others. When the number of users is large, it requires a large number of proposals and corresponding replies from the participating agents to find a commonly available or free date/time slot at which all of them can have the meeting. Each proposal and reply discloses some of their users' private calendar information to other agents. In this way the participating agents learn private calendar information through the negotiation process.

Privacy protection for the MS problem has been achieved in other mechanisms [66] by using costly cryptographic tools. Another objective with the MS problem is to achieve optimality in the solution based on a global objective function, which can be achieved in a Distributed Constraint Optimization (DCOP) algorithm [43, 44, 46]. Distributed algorithms have the advantage of a level of fault tolerance, but suffer from high complexity and high privacy loss.

In this chapter, we present a mobile agent-based scheduling scheme called Efficient and Privacy-aware Meeting Scheduling (EPMS), which has low computational complexity, results in a global utility close to the optimal level necessary, and achieves better privacy protection by utilizing a common computational space.

4.2 Formalization of MS Problem

The meeting scheduling (MS) problem consists of a number of agents, a set of meeting variables associated with the agents, and a set of constraints distributed among these variables, each of which can take values in an associated finite domain. Constraints may exist between the variables of one agent (i.e., intra-agent) or between the variables

of different agents (i.e., inter-agent). Hard constraints represent absolute limitations imposed on the variables and signify what is and is not allowed. Soft constraints can be described via the preferences [27][53]. The instantiation of meeting variables must satisfy all the hard constraints and should try to satisfy the soft constraints as much as possible. We model the MS problem as follows:

1. A set of agents $A = \{A_1, ..., A_N\}$ of cardinality N exists. Each agent A_i, $i \in \{1, ..., N\}$, belongs to one user and is a representative of that user.

2. A meeting set $M = \{M_1, ..., M_K\}$ is the set of all meetings provided by all users. Each meeting M_k, $k \in \{1, ..., K\}$, belongs to a different set $A^k = \{A_1, ..., A_{n_k}\}$ of n_k agents, where $A^k \subset A$ is the subset of participating agents in the meeting M_k. Two meetings M_{k_1} and M_{k_2}, where $k_1, k_2 \in \{1, ..., K\}$, may have one or more common (i.e., $A^{k_1} \cap A^{k_2} \neq \emptyset$) participating agents. The meeting set of agent A_i includes $M^i \subset M$.

3. Each meeting M_k belongs to a set $T^k = \{T_l, T_{l+1}, ..., T_{l+d-1}\}$ of d candidate date/time slots on a calendar. This set T^k is common knowledge among agents A^k of the meeting M_k.

4. A personal calendar T is a sequence of time slots in which some slots are occupied with previously fixed schedules. For simplicity let the slots be of equal duration. A number of slots in the calendar make up the set T^k.

5. Each agent $A_i \in A^k$ has its preference variable set, denoted by $P^{k,i}$, for its free slot set. The value of a preference variable $P^{k,i}_j$, corresponding to $T_j \in T^k$, represents A_i's preference on the date/time slot T_j for the meeting M_k.

6. A preference variable P_j can take a value from a finite domain $V = \{V_1, ...V_m\}$. The domain V is common knowledge among all the participants.

7. The meeting variable set $X = \{X_1, ...X_K\}$ of cardinality K corresponds to the meeting set M, where X_k corresponds to the meeting M_k, $\forall k \in \{1, ..., K\}$. The meeting variable of agent A_i for meeting M_k is denoted by X^i_k.

8. A meeting variable X_k can take a value $T_j \in T^k$. Assigning a value T_j to X_k means that the meeting M_k has been scheduled for time slot T_j. For simplicity, let the meeting length be one time slot.

9. Intra-agent constraints exist among the meeting variables of the same agent. For any pair of meetings, M_{k_1} and M_{k_2}, that belong to at least one common agent $A_c \in (A^{k_1} \cap A^{k_2})$, the corresponding meeting variable values cannot be equal. Inter-agent constraints exist between variables of different agents. For any two agents A_{i_1} and A_{i_2}, the corresponding meeting variables for the same meeting must be equal.

10. The intra-agent constraints of agent A_i on X_k^i are the private information of agent A_i and known only to agent A_i. The inter-agent constraints between $X_k^{i_1}$ and $X_k^{i_2}$, which belong to A_{i_1} and A_{i_2} respectively, are distributed between agents A_{i_1} and A_{i_2}.

11. The values assigned to the preference variables $P_j^{k,i}, \forall j \in \{l, l+1, ..., l+d-1\}$, $\forall k \in \{1, ..., K\}$ are the private information of agent A_i.

12. The local utility of a meeting is the sum of the preference values of all agents at slot T_j for which that meeting is scheduled (i.e. $\sum_{A_i \in A^k} \sum_{T_j \in T^k} P_j^{k,i}$) and the global utility is the sum of the local utilities of all the meetings.

13. A feasible solution S is an instantiation of the meeting variable set to the time domain that satisfies all the constraints, where $S(M_k) \subset T$ is the time slot assigned for meeting M_k. An optimal solution is an instantiation of all the meeting variables such that the global utility $\sum_{k=1}^{K} \sum_{A_i \in A^k} \sum_{T_j \in S(M_k)} P_j^{k,i}$ is maximized.

4.3 Proposed Efficient and Privacy-aware Meeting Scheduling (EPMS) Scheme

EPMS uses a common computational space on the agent server, which is shared among the participants of the same meeting. Like the blackboard model [13], it is used to aggregate the agents' data into a single space. However, unlike the blackboard model,

there is no monitor on the common computational space and the sequence of actions of the agents in it is random and independent of the other agents' actions. Thus, the relationship of the private information (used in the common computational space) with the specific individual remains secret to the participants, resulting in better privacy. It is not used to monitor and trigger agents' actions as in the blackboard model; rather it is utilized for reducing privacy loss.

A common computational space can be a shared file that gives access to only the participants of the same meeting. An organization may install and maintain its own agent server to be used for scheduling meetings among its members. Members of different organizations may use a server from a third party service provider. In any case, the agent server should be configured according to trusted computing specifications [36].

An organization may install and maintain its own agent server to be used for scheduling meetings among its members. Members of different organizations may use servers from third party service providers. We will consider the scheduling of a single meeting first and then the scheduling of multiple meetings. Finding the proper schedule for a meeting is to find a common date/time slot from the candidate slots where all the participants are free and the mean user preference is maximized.

4.3.1 Basic Algorithm

The mobile agent-based meeting scheduling scheme EPMS has four phases. For scheduling a meeting M_k, in the first phase, an initiating agent invites all other participants and informs them about the meeting parameters, like the candidate date/time slot set T^k, the participant set A^k, the meeting length, and some common computational spaces. For simplicity, let us assume the meeting length to be one time slot. Each participating mobile agent, along with its corresponding user's private calendar information and user preferences on the free slots, migrate into the agent server. This migration is controlled by the access control mechanisms in the agent server. In the second phase, each agent $A_i \in A^k$ finds its date/time slot set $T_z^{k,i} \subset T^k$ in its personal calendar that is already occupied with previously fixed schedules and the slot set $T_a^{k,i} \subset T^k$ in which it is available or free. Then, each agent encodes its constraints

into a constraint vector C^k of size d (where, d is the cardinality of T^k) based on its unavailability and preferences in the free slots. In the third phase, each agent participates in constructing a combined utility factor vector UF^k of size d at the common computational space synchronously, but in random sequence. If any agent has an intra-agent hard constraint related to a specific date/time slot $T_j \in T_z^{k,i}$, for which it cannot attend a new meeting at that date/time slot, the UF of that slot UF_j^k is made zero; otherwise its preference value $P_j^{k,i}$ on that slot is added with the UF_j^k. After all of the member agents participation, the UF of all unavailable slots UF_z^k will become zero and that of all available slots UF_a^k will have non-zero values reflecting group preferences or utility at each slot.

The combination of the constraints of all agents into the UF vector gives a picture of the overall preferences on individual slots of the commonly available slot set $T_a^k \subset T^k$ and also shows all unavailable slot set $T_z^k \subset T^k$. All non-zero valued slots T_a^k in the UF vector are feasible solutions, since all of the members can attend the meeting at any of them. After constructing the combined UF vector, in the fourth phase, the agents find slot $T_m \in T_a^k$ at which the UF_m^k is maximized and assign $X_k = T_m$.

4.3.2 Encoding Constraints

The constraint vector C^k created by agent A_i is denoted by $C^{k,i}$ whose j^{th} element is defined as follows.

$$C_j^{k,i} = \begin{cases} 0; & if\ T_j \in T_z^{k,i} \\ P_j^{k,i}; & if\ T_j \in T_a^{k,i} \end{cases} \tag{4.1}$$

In this equation, $0 < P_j^{k,i} \leq 1$ is the preference value at time slot T_j.

Table 4.1: Examples of encoded constraint vectors for five agents.

	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
$C^{k,1}$	1.00	0	0.45	0.70	0	0.80	0.20	0	0.50
$C^{k,2}$	0.75	0.90	1.00	0	0.50	0.65	0.60	0.40	0.55
$C^{k,3}$	0.55	0.30	0.80	0	0	0	0.70	1.00	0.30
$C^{k,4}$	0.25	0.40	0.90	0.80	0.20	0	0.35	1.00	0.65
$C^{k,5}$	0.50	0	0.35	0	0	0	0.30	1.00	0.45

Between two separate slots, the higher valued slot is considered more preferred

than the lower valued slot. Table 4.1 shows an example of the constraint vectors $C^{k,1}$, $C^{k,2}$, $C^{k,3}$, $C^{k,4}$, and $C^{k,5}$ created for M_k by the participating agents A_1, A_2, A_3, A_4, and A_5, respectively. From the table, agent A_1 has intra-agent hard constraints for which it cannot attend a new meeting at any of the T_2, T_5, and T_8 slots. Among its free slots it prefers T_1 the most, then T_6, and so on. Similarly, other agents also have hard constraints for some slots and preference values for their free slots.

4.3.3 Utility Factor

After independently encoding constraints into a constraint vector, each agent participates in building the combined UF vector of the meeting in random order. Initially, the UF vector contains null values. Each agent A_i updates its j^{th} element UF_j^k $\forall j \in \{l, l+1, ..l+d-1\}$ according to the following equation,

$$UF_{j(s)}^k = \begin{cases} 0; & if\ C_j^{k,i} = 0\ or\ UF_{j(s-1)}^k = 0 \\ UF_{j(s-1)}^k + C_j^{k,i}; & otherwise \end{cases} \tag{4.2}$$

where, s is the sequence number of agent A_i in constructing the UF^k vector and $UF_{j(s)}^k$ is the partial UF at slot T_j, calculated by A_i in sequence number s. For instance, let the UF vector construction sequence be A_4, A_3, A_5, A_1, and A_2. With the constraint vectors given in Table 4.1, the partial UF vectors built by the participants will be as shown in Table 4.2.

After all member's have participated, the UF vector created by the last agent in the construction sequence (e.g., $UF_{2(5)}$ in Table 4.2) is the combined UF vector.

4.3.4 Search Procedure

The combined UF vector for a meeting may have some non-zero valued slots (e.g., $T_a^k = \{T_1, T_3, T_7, T_9\}$ in Table 4.2 (e)) indicating that all of the participants are free in those slots. All these available slots T_a^k are feasible solutions to the problem. The utility factor value at slot T_j expresses the utility and the overall preference value of the participants in the T_j slot. To get the maximum utility, an agent performs a linear search for slot T_m in T_a^k for which UF_m^k is maximized and assigns to $X_k = T_m$. In the example in Table 4.2, $X_k = T_3$.

Table 4.2: Construction of combined utility factor vector.

(a)Partial UF vector $UF^k_{(1)}$ after A_4's participation

T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
0.25	0.40	0.90	0.80	0.20	0	0.35	1.00	0.65

(b)Partial UF vector $UF^k_{(2)}$ after A_3's participation

T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
0.80	0.70	1.70	0	0	0	1.05	2.00	0.95

(c)Partial UF vector $UF^k_{(3)}$ after A_5's participation

T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
1.30	0	2.05	0	0	0	1.35	3.00	1.40

(d)Partial UF vector $UF^k_{(4)}$ after A_1's participation

T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
2.30	0	2.50	0	0	0	1.55	0	1.90

(e)Complete UF vector UF^k after A_2's participation

T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
3.05	0	3.50	0	0	0	2.15	0	2.45

4.3.5 Multiple Meetings

For scheduling multiple meetings concurrently, the construction of the UF vectors for each meeting can be done simultaneously in the same process as described in Sections 4.3.2 to 4.3.4. However, they cannot be scheduled without synchronization because one or more agents that participate in more than one of these meetings may exist. The scheduling priority SP_k of meeting M_k is determined by the maximum value in UF^k according to the following equation.

$$SP_k = Max(UF^k_j), \forall T_j \in T^k \tag{4.3}$$

For any two meetings M_{k_1} and M_{k_2} that have at least one common agent $A_c \in (A^{k_1} \cap A^{k_2})$, the meeting with the higher scheduling priority, say M_{k_2}, is scheduled first, say for slot T_{m_2} at which $UF^{k_2}_{m_2}$ is maximized. Then, each common agent A_c updates the UF value of the other meeting in slot T_{m_2} (i.e., $UF^{k_1}_{m_2}$) to zero if T_{m_2} is also a candidate slot for the meeting M_{k_1}. This update is necessary because a new intra-agent constraint ($X^c_{k_1} \neq X^c_{k_2}$) has arisen due to the newly scheduled meeting M_{k_2}. Then, meeting members find the slot T_{m_1} for M_{k_1} at which $UF^{k_1}_{m_1}$ is maximized.

Figure 4.1 shows an example of the sets of agents A^k for five meeting and Table

4.3 shows the respective utility factor vectors of those meetings in which the sets of candidate slots for each meeting are shaded. There are common agents for two or more meetings (e.g., $A^3 \cap A^5 = \{A_4, A_{14}\}$. Also, there are meetings with overlapping candidate slots (e.g., $T^3 \cap T^5 = \{T_3, T_4, T_5, T_6, T_7, T_8\}$).

$A^1 = \{A_6, A_{12}, A_{15}, A_{19}\}$
$A^2 = \{A_2, A_4, A_7, A_{13}, A_{14}\}$
$A^3 = \{A_1, A_4, A_9, A_{14}\}$
$A^4 = \{A_1, A_{13}, A_{15}\}$
$A^5 = \{A_3, A_4, A_8, A_{10}, A_{14}\}$

Figure 4.1: Example of the sets of agents for five meetings.

Table 4.3: Utility factor vectors for five meetings.

	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}	T_{12}
M_1	0	2.4	3.1	0	2.9	0						
M_2								3.5	0	0	2.9	0
M_3			2.1	1.4	2.6	0	0	0.6	2.8	1.7		
M_4							1.9	2.4	1.3	0	1.2	
M_5			4.2	0	0	0	2.5	3.1				

According to Table 4.3, the scheduling priorities of the meetings are $SP_1 = 3.1, SP_2 = 3.5, SP_3 = 2.8, SP_4 = 2.4$, and $SP_5 = 4.2$. Thus, M_5, which has the highest scheduling priority, is scheduled first at the highest utility valued slot T_3, i.e., $X_5 = T_3$. But $A^5 \cap A^2 = \{A_4, A_{14}\}$ and $A^5 \cap A^3 = \{A_4, A_{14}\}$. However, since $T_3 \notin T^2$, agents A_4 and A_{14} do not update UF_3^2 with zero and since $T_3 \in T^3$, agents A_4 and A_{14} update UF_3^3 with zero. Although $T_3 \in T^1$, there are no common agents between them i.e., $A^1 \cap A^5 = \emptyset$. So, UF_3^1 is not updated by any agent after scheduling M_5.

The scheduling of the highest priority meeting and updating of the UF vectors of the remaining meetings are carried out until all meetings are scheduled. In the example shown in Fig. 4.1 and Table 4.3, the meetings will be scheduled as $X_1 = T_3$, $X_2 = T_8$, $X_3 = T_9$, $X_4 = T_7$, and $X_5 = T_3$. Note that M_5 and M_1 both are scheduled at T_3 because T_3 has the maximum utilities for both of them and $A^1 \cap A^5 = \emptyset$. Figure

4.2 presents the EPMS algorithm.

01: when received invite(T^k, A^k) do //creating C^k
02: add M_k to M^i
03: **for** $\forall T_j \in T_a^k$ **do**
04: **if** free at T_j
05: $C_j^k \leftarrow P_j^k$;
06: **else**
07: $C_j^k \leftarrow 0$
08: **end if**
09: **end for**
10: *call* migration(server address);
11: procedure buildUF(k) //procedure for building UF^k
12: **while** $updating\_flag(k) == true$ **do**
13: wait (random());
14: **end while**
15: $updating\_flag(k) \leftarrow true$
16: **for** $\forall T_j \in T^k$ **do**
17: **if** $C_j^k == 0 || UF_j^k == 0$ **do**
18: $UF_j^k \leftarrow 0$;
19: **else**
20: $UF_j^k \leftarrow UF_j^k + C_j^k$;
21: **end if**
22: **end for**
23: $updating\_flag(k) \leftarrow false$
24: procedure calculateSP(k) //calculating SP_k
25: $SP_k \leftarrow 0$; $max \leftarrow 0$;
26: **for** $\forall T_j \in T^k$ **do**
27: **if** $UF_j^k > max$ **do**
28: $max \leftarrow UF_j^k$;
29: **end if**

30: **end for**
31: $SP_k = max$;
32: procedure schedule(k) //scheduling k^{th} meeting
33: $max \leftarrow 0; m \leftarrow -1$;
34: **for** $\forall T_j \in T^k$ **do**
35: **if** $UF_j >= max$ **do**
36: $max \leftarrow UF_j$; $m \leftarrow j$;
37: **end if**
38: **end for**
39: **if** $m >= 0$ **do**
40: $X_k = T_m$;
41: **end if**
42: **for** $\forall M_{k_1 \neq k} \in M^i$ **do**
43: **if** $T_m \in T^{k_1}$ **do**
44: **while** $updating\_flag(k_1) == true$ **do**
45: wait (random());
46: **end while**
47: $updating\_flag(k_1) \leftarrow true$
48: $UF_m^{k_1} \leftarrow 0$;
49: $updating\_flag(k_1) \leftarrow false$
50: **end if**
51: **end for**
52: remove M_k from M^i
53: procedure schedule(M^i) //scheduling all meetings M^i of A_i
54: **for** $\forall M_k \in M^i$ **do**
55: *call* buildUF(k);
56: *call* calculateSP(k);
57: **end for**
58: *call* sort (M^i, SP^i);//arrange M^i based on SP^i order
59: **for** $\forall M_k \in M^i$ **do**
60: *call* schedule(k)

61: **end for**

Figure 4.2: EPMS algorithm for scheduling multiple meetings.

We considered the utility as the sum of user preferences and accordingly, the scheduling priority of a meeting in scheduling multiple meetings was taken as the maximum utility. Our objective function was to maximize the utility. Sometimes the scheduling cost is taken into account and the objective function is to minimize the cost and sometimes both the cost and the utility are taken into account and accordingly the objective function is to maximize utility and minimize the cost. Based on the objective function, the scheduling priority in EPMS must be changed. For example, if the cost is included in the objective function, then the scheduling sequence in EPMS would be determined by the maximum value of the ratio of the sum of the preferences to the sum of the costs, instead of the maximum value of the sum of the preferences.

4.4 EPMS Characteristics

In this section we analyze the EPMS algorithm to measure its complexity, privacy loss, and global utility (GU) in scheduling multiple meetings concurrently.

4.4.1 Complexity

In EPMS algorithm, for each meeting $M_k \in M$ each of the n_k participants encodes their own constraints into a d_k sized constraint vector C^k, where d_k is the cardinality of the candidate date/time slot set T^k and n_k is the number of participants in the meeting M_k. This encoding requires maximum $\text{Ç}_1 = \sum_{k=1}^{K}(n_k \times d_k)$ key operations by all agents of all meetings. Then, for building the combined utility factor vector UF^k, each participant compares and updates the d_k element data at the common computational space, which needs maximum $\text{Ç}_2 = \sum_{k=1}^{K}(n_k \times d_k)$ key operations for all meetings. Next, for calculating the scheduling priorities for all meetings, a maximum of $\text{Ç}_3 = \sum_{k=1}^{K}(n_k \times d_k)$ key operations are necessary. Before scheduling multiple meetings, an agent arranges the meeting set M^i based on their scheduling priorities' descending

order, which requires maximum $Ç_4 = \sum_{i=1}^{n} m_i^2$ operations, where n is the number of participants in all meetings and m_i is the number of meetings in which agent A_i is a participant. While scheduling each meeting M_k, each agent A_i performs a linear search operation in the UF^k vector to find a slot having the maximum UF value, which requires maximum $Ç_5 = \sum_{k=1}^{K}(n_k \times d_k)$ number of key operations for all meetings. Finally, after scheduling a meeting M_k, each of its participants A_i updates the UF vector at slot T_k for its remaining meetings, if any. An agent A_i updates maximum $(m_i - 1) + (m_i - 2) + ... + 1 = m_i(m_i - 1)/2$ slots for all of its meetings. The maximum update operation by all agents is $Ç_6 = \sum_{i=1}^{n}(m_i(m_i - 1)/2)$. Combining all key operations, the total number of key operations by all agents for all meetings becomes

$$Ç = 4\sum_{k=1}^{K}(n_k \times d_k) + \sum_{i=1}^{n}\frac{m_i(3m_i - 1)}{2}. \tag{4.4}$$

If we assume that all of the agents participate in all meetings (i.e., $n_{k_1} = n_{k_2} = n, \forall k_1, k_2 \in K$ and $m_{i_1} = m_{i_2} = K, \forall A_{i_1}, A_{i_2} \in A$) and all meetings have the same number of candidate slots (i.e., $d_{k_1} = d_{k_2} = d, \forall k_1, k_2 \in K$), then $Ç = 4 \times K \times n \times d + n \times K(3K - 1)/2 = K \times n(4d + (3K - 1)/2)$. So, the worst case complexity for the EPMS algorithm is of the order $O(Knd)$. If one meeting is scheduled in each of the candidate slots (i.e., $K = d$), then the complexity of EPMS becomes $O(nd^2)$ i.e., polynomial.

4.4.2 Privacy Analysis

Existing privacy evaluation techniques for an MS algorithm take into consideration only one criterion (usually preference) [25][42]. But we think that both the user's personal schedule and preferences at different candidate slots should be considered private information. Thus, we introduce multiple criteria, which considers both the schedule privacy and preference privacy, for evaluating privacy in an MS algorithm.

Schedule Privacy

For the schedule privacy loss measurement of agent A_i, we need to find out the number of schedule states of A_i in the candidate slot span that other agents can figure out.

An agent can be either busy, say an $s1$ state, or free, say an $s2$ state, for any specific time slot (i.e., $S_i = \{s1, s2\}$) and the schedule states of an agent at a slot are mutually exclusive. The VPS privacy metric of agent A_i in meeting M_k for its personal schedule can be represented as,

$$\begin{aligned}\mathbb{V}_i(\mathbb{P}_i(S_i)) &= \sum_{m \neq i} \sum_{T_{j_1} \in T_z^k} I_{\{\mathbb{P}_i^m(s_i = s1(T_{j_1}))\}} + \\ &\sum_{m \neq i} \sum_{T_{j_2} \in T_a^k} I_{\{\mathbb{P}_i^m(s_i = s2(T_{j_2}))\}} \end{aligned} \tag{4.5}$$

where, T_z^k and T_a^k are the sets of unavailable (i.e., zero valued) slots and available (i.e., non-zero valued) slots in the UF vector, respectively. Once the value of a slot of the UF vector UF_j is set to zero, no other agent changes its value. Thus, a zero valued slot confirms one agent's busy state ($s1$). Since, the UF vector is created by multiple agents in a random sequence, the probability of characterizing agent A_i with the busy state ($s1$) at a zero valued slot $T_{j_1} \in T_z^k$ by an observing agent A_m is $\mathbb{P}_i^m(s_i = s1(T_{j_1})) = 1/(n-1)$. On the other hand, the probability of characterizing an agent A_i with the free state ($s2$) at a non-zero valued slot $T_{j_2} \in T_a^k$ by an observing agent A_m is $\mathbb{P}_i^m(s_i = s2(T_{j_2})) = 1$, since all of the agents are free at T_{j_2}. The states of A_i revealed to each observing agent are the same, because the probability function $\mathbb{P}_i(S_i)$ is the same for each observing agent. So, the VPS privacy metric for a personal schedule is,

$$\mathbb{V}_i(\mathbb{P}_i(S_i)) = (n-1) \times (\frac{z}{n-1} + (d-z)) \tag{4.6}$$

where, z is the cardinality of $T_z^k \subset T^k$ representing the number of unavailable slots and $(d-z)$ is the cardinality of $T_a^k \subset T^k$ representing the number of free slots. Note that $T_z^k \cup T_a^k = T^k$. In the VPS privacy metric, an observed agent has $((n-1) * d)$ schedule states to the observing agents and potentially a total of one unit of privacy to lose. Thus, the value of α is equal to $(1/((n-1) * d))$ and A_i's schedule privacy loss is,

$$\mathcal{P}_i(s) = \frac{1}{d} \times (\frac{z}{n-1} + (d-z)). \tag{4.7}$$

The Min privacy metric does not account for the number of observing agents to whom the states are revealed. So, the Min privacy metric for a personal schedule is,

$$\mathbb{V}_i(\mathbb{P}_i(S_i)) = \frac{z}{n-1} + (d-z). \tag{4.8}$$

In the Min privacy metric, an observed agent has a total d number of states to lose to the observing agents. If all of the d states are revealed to the observing agents, the privacy loss is one i.e., $\alpha = 1/d$. Thus, in the Min privacy metric, A_i's schedule privacy loss $\mathcal{P}_i(s)$ is,

$$\mathcal{P}_i(s) = \frac{1}{d} \times (\frac{z}{n-1} + (d - z)). \tag{4.9}$$

The schedule privacy loss in EPMS is the same in the two metrics and high for a small number of participants. In a worst case scenario, for two participants, the schedule privacy loss is one. This is determined from Eq. (4.7). For a fixed value of d, if the number of participants in a meeting is increased, the number of zero-valued slots z in the UF vector will increase and the non-zero-valued slots $(d - z)$ will decrease; as a result the schedule privacy loss will decrease, and the privacy level will increase.

Preference Privacy

The preference privacy loss measures the number of preference states of an observed agent revealed to its observing agents. If the observing agents can accurately determine all of the preference values (i.e., states) of agent A_i, then the privacy loss of A_i will be one. If they cannot determine any of its preference values, the privacy loss will be zero. The participating agents in EPMS do not exchange their preferences for different slots amongst themselves. While constructing the combined UF vector, each agent reads existing values and updates them by adding its own preference values for each slot. Thus the preference values of all agents get modified in the UF vector. When the agents read the existing values from the UF vector, each of them, except for the first two agents in the construction sequence, reads sum of the preferences of the previous agents. They cannot split them up into values that were summed up. But, the first agent reads no other agent's preference values and the second agent reads the preference values of only the first agent.

Suppose agent A_i has p number of free slots and a preference value for each of its free slots in UF^k. In EPMS, out of n agents, only one agent's (i.e. the first agent)preference values are revealed to only one observing agent (i.e. the second agent). But agent A_i's probability of being the first agent in the construction sequence is $1/n$. So, on the average, each agent A_i loses p/n preference states to all of the

observing agents. Thus, the VPS privacy metric for preferences becomes,

$$\mathbb{V}_i(\mathbb{P}_i(S_i)) = \frac{p}{n}. \tag{4.10}$$

Agent A_i has $((n-1)*p)$ preference states to the observing agents. Similar to the schedule privacy loss, the value of α in the VPS privacy metric is $\alpha = 1/((n-1)*p)$ and A_i's preference privacy loss is,

$$\mathcal{P}_i(p) = \frac{1}{n(n-1)}; \qquad n > 2. \tag{4.11}$$

Since, an agent A_i loses p/n preference states to the observing agents, the Min privacy metric for the preferences becomes,

$$\mathbb{V}_i(\mathbb{P}_i(S_i)) = \sum_{s_i \in S_i} \sum_{j=1}^{p} I_{\{\mathbb{P}_i^{m \neq i}(s_i(T_j))\}} = \frac{p}{n}. \tag{4.12}$$

In the Min privacy metric, agent A_i has a total p number of preference states to the observing agents. If agent A_i loses its p number of preference states to the observing agents, its preference privacy level will be zero. So, $\alpha = 1/p$ and A_i's preference privacy loss is,

$$\mathcal{P}_i(p) = \frac{1}{n}; \qquad n > 2. \tag{4.13}$$

In the special case of only two participants, the second agent reads the first agent's preferences and the first agent can also deduce the second agent's preferences by subtracting its own preference values from the combined UF vector. So, for two participants, the preference privacy loss in EPMS is one in both metrics.

4.4.3 Global Utility

In scheduling a single meeting with EPMS, the agents always find the slot with maximum utility among all the available slots for that meeting. In scheduling multiple meetings, the global utility (GU) obtained in the solution depends upon the scheduling sequence, because a newly scheduled meeting creates new constraints on the remaining meetings; each scheduled meeting may change the remaining meetings' possible solution sets, and thus their utility may change. A simple way of gaining optimal GU is to calculate the GU of all possible scheduling sequences (i.e., sequence

permutations) and then finding the sequence that gives the maximum GU. However, this approach is not feasible in terms of computational complexity. An optimization algorithm, such as optAPO [44], can always find the optimal solution. EPMS is not an optimization algorithm. The global utility in EPMS will be less than the optimal. We compared the GU achieved in EPMS with that of an optimization algorithm using simulation experiments.

4.5 Experiments and Analysis

4.5.1 Experimental Setup

For evaluation and analysis of the privacy loss and global utility in the EPMS algorithm, we carried out simulation experiments written in Java. We chose a maximum of $K = 10$ new meetings to be scheduled concurrently, among a maximum of $N = 20$ agents and with a total of $D = 40$ time slots. For different meetings, we used random values for n from a range of 3 to 20 and random values for d from a range of 5 to 15. The candidate slots of a meeting were adjacent and started at a random position between the 1^{st} slot and the $(D - d + 1)^{th}$ slot. For each agent, a random percentage of slots, ranging from 5% to 50% of the candidate slots, were considered already occupied with previously fixed schedules and distributed at random positions within the candidate slots. In the free slots, non-zero random preference values ($0 < p_i \leq 1$) were set. The simulation was run 400 times for each experiment, and the average values for the runs were taken. Unless otherwise stated for any other values for an input parameter, the above-mentioned values should be considered.

EPMS uses mobile agents while distributed algorithms use stationary agents. The mobile agents in EPMS need to migrate to the agent server and the stationary agents in distributed algorithms need to exchange remote messages. We also measured the mobile agent migration cost/overhead and stationary agent messaging cost/overhead and compared the two systems. For this experiment, we used an open source agent development kit (Aglet Ver. 2.0.2) [39].

4.5.2 Experimental Results

For a fixed number of candidate slots d, if the number of participants n in a meeting increases, the probability of having an intra-agent constraint at a specific slot increases, i.e. $(d - z)$ decreases towards zero. Figure 4.3 shows the number of commonly available slots for three different values, $d = 5$, $d = 10$, and $d = 15$, for varying numbers of participants n.

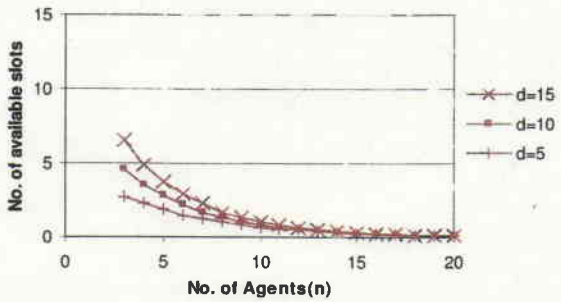


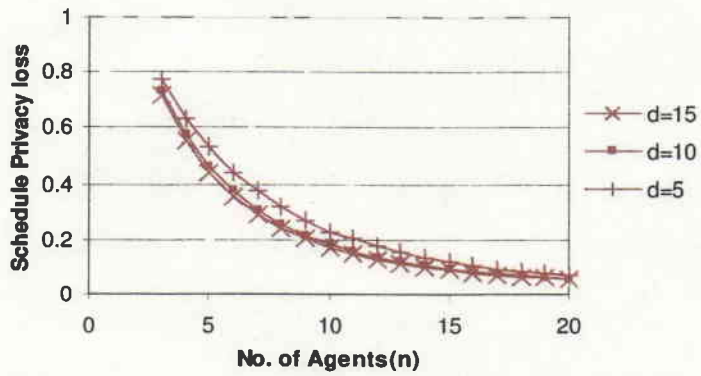
Figure 4.3: Number of commonly available slots for varying number of participants.

From Fig. 4.3 we see that, with the increase in the number of candidate slots, the increase in the number of available slots is more for smaller numbers of participants than that for larger numbers of participants.

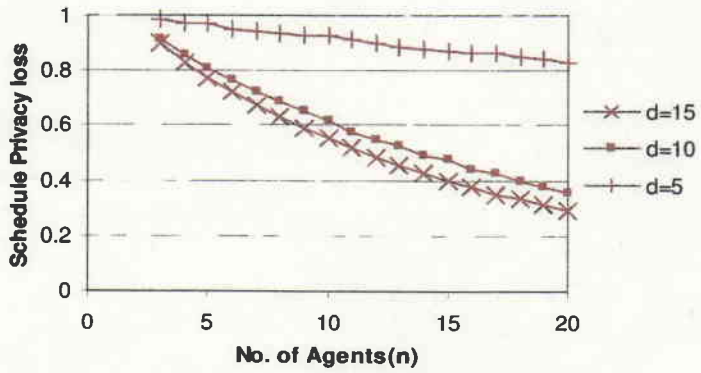
We compare the privacy loss $\mathcal{P}_i$ in EPMS algorithm with those of two other algorithms- centralized algorithm, which has a low privacy loss in the VPS metric [42] and optAPO [44], which is a well-known optimization algorithm. For the centralized algorithm, we consider that one of the participants takes all of the inputs and solve the problem by using a centralized algorithm. The privacy loss of an agent in optAPO depends upon how many neighbors the agent has. For the purpose of analysis, we show the minimum privacy loss for optAPO, which occurs for cases where each agent discloses all its private information to only one neighbor in the initialization phase.

Schedule Privacy Loss

The schedule privacy loss $\mathcal{P}_i(s)$ of an agent A_i depends upon the number of unavailable slots z, and commonly available slots $(d - z)$ among the candidate slots in the UF vector as per equation(14).



(a) Over-constrained.



(b) Under-constrained.

Figure 4.4: Schedule privacy loss in EPMS for varying number of participants.

Figure 4.4(a) shows the schedule privacy loss $\mathcal{P}_i(s)$, in the VPS privacy metric, of an agent A_i in EPMS for different values of $d = 5$, $d = 10$ and $d = 15$ with varying number of participants in over-constrained settings (occupancy with formerly scheduled events is 5% to 50% of candidate slots) and Fig. 4.4(b) shows for under-constrained settings (occupancy with formerly scheduled events is 0% to 20% of candidate slots). In real life, the level of constraints of a participant in the candidate date/time slots is low if the candidate date/time slot is far from the current date.

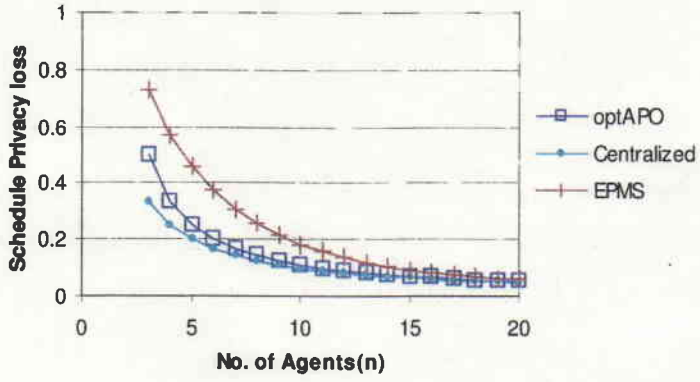
From Fig. 4.4(a) we see that for small number of participants n, the schedule privacy loss of an agent is high. For three participants the loss is above 0.7. But if the number of participants n is increased, the schedule privacy loss of an agent gets decreased. The schedule privacy loss in under-constrained settings is more than that in

over-constrained settings because of the increase in the number of commonly available slots $(d - z)$. We also see that when the number of candidate slots d for a meeting is increased, the schedule privacy loss gets decreased more in under-constrained settings than in over-constraint settings.

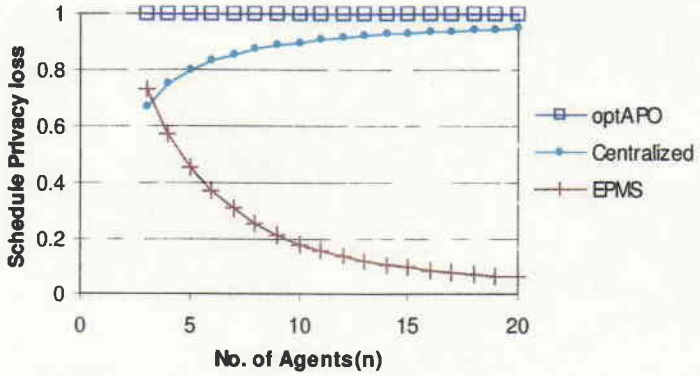
In the VPS privacy metric, each agent in optAPO has $((n-1) * d)$ states to the observing agents and in the best case (i.e., one neighbor) each agent loses its d states to its neighbors. So, its schedule privacy loss in the VPS privacy metric is $\mathcal{P}_i(s) = d/((n-1) * d) = 1/(n-1)$. On the other hand in the centralized algorithm, $(n-1)$ agents (out of n agents) give away their d number of schedule states to the central agent making the average number of states revealed per agent $((n-1) * d/n)$. In the VPS privacy metric, each agent has $((n-1) * d)$ states to the observing agents. So, the schedule privacy loss of an agent in the centralized algorithm is $\mathcal{P}_i(s) = ((n-1) * d/n)/((n-1) * d) = 1/n$. Figure 4.5(a) compares the scheduling privacy loss (for $d = 10$) in EPMS with two other algorithms in the VPS privacy metric.

We see that, in the VPS privacy metric the schedule privacy loss in the centralized algorithm is less than that in EPMS and optAPO. This is because, in addition to the number of participants who lose their information, the VPS privacy metric accounts for the ratio of the number of observing agents to whom the information is revealed to the total number of observing agents. Generally speaking, in the centralized algorithm, the entire schedule information of an agent is revealed to only one observing agent while in EPMS algorithm, part of the schedule information is revealed equally to all observing agents through inference on the feasible solution set. The privacy loss in optAPO algorithm shown here is for the case in which each agent has only one neighbor. The more number of neighbors in optAPO, the more privacy loss in VPS privacy metric. The loss will be doubled if the agents have 2 neighbors on an average.

In the Min privacy metric, each of $(n-1)$ agents in the centralized algorithm lose its d number of states to the observing agents making the average number of states revealed per agent $((n-1) * d/n)$. Each agent has d states to the observing agents. So, the schedule privacy loss of an agent in the centralized algorithm is



(a) VPS privacy metric



(b) Min privacy metric

Figure 4.5: Relative schedule privacy loss in three algorithms for varying number of participants.

$\mathcal{P}_i(s) = ((n-1)*d/n)/d = (n-1)/n$. In optAPO each agent loses its entire schedule to its neighbors. So, the schedule privacy loss in optAPO in the Min privacy metric is one. Figure 4.5(b) compares the scheduling privacy loss in EPMS algorithm with that of two other algorithms in the Min privacy metric. We see that in the Min privacy metric, the privacy loss in EPMS is much less than the two other algorithms.

From Fig. 4.5(a) and 4.5(b), the privacy loss in the centralized algorithm decreases in the VPS privacy metric while it increases in the Min privacy metric with the increase in n. This is due to the differences in the definition of the two metrics. The Min privacy metric accounts for the number of participants that lose their information to any observing agents, regardless of how many observing agents it is revealed to. In the centralized algorithm, $(n-1)$ agents lose their information to other (i.e.,

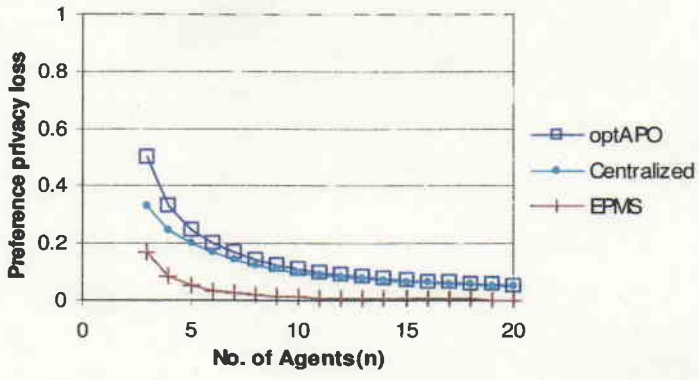
central) agents. Thus, with an increase in n, the privacy loss increases in the Min privacy metric (Fig. 4.5(b)). But, when the privacy metric includes the fraction of the observing agents to whom the information is lost, as in the VPS privacy metric, the privacy loss in the centralized algorithm becomes much less (Fig. 4.5(a)), since the loss occurs only to one observing agent. The schedule privacy loss in optAPO is one in the Min privacy metric. But in the VPS privacy metric, the loss in optAPO is much less for the same reason as described above for the centralized algorithm. In EPMS, the schedule privacy loss does not change in the VPS privacy metric because the information is revealed to all observing agents.

Preference Privacy Loss

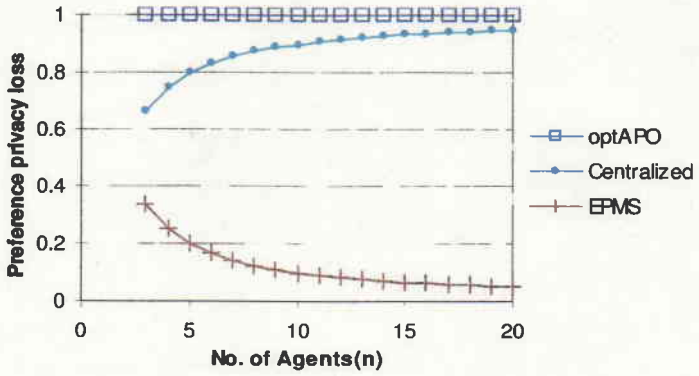
An agent has p number of preference values in its free slots. Similar to the schedule privacy loss, in optAPO, at least p states are revealed to at least one neighbor. So, the preference privacy loss $\mathcal{P}_i(p)$ in the VPS privacy metric is $\mathcal{P}_i(p) = p/((n-1) * p) = 1/(n-1)$ and that in the Min privacy metric is $\mathcal{P}_i(p) = p/p = 1$. In the centralized algorithm, all the agents, except the central agent (i.e., $(n-1)$ agents out of n agents), give away their p number of preference states to the central agent making the average number of states revealed per agent $(n-1) * p/n$. So, in the centralized algorithm, the preference privacy loss of an agent in the VPS privacy metric is $\mathcal{P}_i(p) = ((n-1) * p/n)/((n-1) * p) = 1/n$ and that in the Min privacy metric is $\mathcal{P}_i(p) = ((n-1) * p/n)/p = (n-1)/n$.

The graph in Fig. 4.6(a) shows the preference privacy loss $\mathcal{P}_i(p)$ for three algorithms, in the VPS privacy metric and Fig. 4.6 (b) shows the same information in the Min privacy metric, for varying number of participants n. EPMS outperforms both the centralized algorithm and optAPO for any number of participants and its preference privacy loss is very low. Also, in the Min privacy metric, the preference privacy loss in EPMS is much less than for the other two algorithms. The preference privacy loss is independent of the number of candidate slots and the level of constraints when it is normalized for [0..1] scale.

Unlike schedule privacy loss, the preference privacy loss in EPMS is less in the VPS privacy metric than in the Min privacy metric. This is because unlike the schedule



(a) VPS privacy metric



(b) Min privacy metric

Figure 4.6: Relative preference privacy loss in three algorithms for varying number of participants.

information, the same preference information is not revealed to all observing agents. Only one agent's preference information is revealed to only one observer. Thus, in the VPS privacy metric, which accounts for a fraction of the observing agents to whom the information is revealed, the average loss becomes less than that in the Min privacy metric.

Unlike the traditional algorithms, the agents in EPMS do not exchange their private information directly with one another; they update the shared UF vector with their personal information in a random sequence. Thus, the probability of mapping ($\mathbb{P}_m$) the data, found in the utility factor vector, with a specific agent A_i is less than one. As a result, the privacy loss in EPMS is less than those in the centralized algorithm and distributed algorithms.

Global Utility

Distributed Constraint Optimization (DCOP) algorithms (e.g., OptAPO [44]) give the optimal global utility (GU) for solving distributed constraint satisfaction problems. Since all optimization algorithms give the same (optimal) GU, we use the legend "Opt" to represent a DCOP algorithm. To show EPMS performance, we measure global utility of EPMS and compare it with that of a DCOP algorithm for various input parameters. We took the sum of preferences at the scheduled slot of a meeting as its local utility and the global utility is the sum of local utilities of all meetings. If a meeting cannot be scheduled, its local utility becomes zero. So the GU value depends upon the number of successfully scheduled meetings. In our experiments, we found that the average scheduling success ratio of a meeting decreases sharply with the increase in the number of participants and EPMS has almost the same scheduling success ratio as in DCOP algorithms.

Figure 4.7 shows the average scheduling success ratio of a meeting in EPMS and the optimal solutions with over-constraint settings for varying n. The graph shows the average success ratio for seven meetings in 400 runs. We also measured the average scheduling success ratio for four and five meetings and they were found to be very close to that of seven meetings. To better visualize the relative success ratio of EPMS and optimal solution, we do not put all data in Fig. 4.7. From the graph, we see that the success ratio in EPMS is very close to the DCOP algorithms. The more the number of participants, the more constraints at a time slot and the less chance of finding a commonly available slot for the meeting. When the number of participants increases, less number of meetings can be scheduled.

Figure 4.8 shows the GU in EPMS and in the optimal solution for varying numbers of participants n. Initially, the GU increases with the increase in the number of participants n, since each participant increases the local utility. However, after a certain value of n, the GU starts decreasing because of the decrease in the number of successful scheduling. From Fig. 4.8, the GU of EPMS is very close to the optimal solution for smaller and larger numbers of participants. However, the difference between them increases near the value of n that gives a maximum GU in the optimal solution. Depending upon the objective function (whether cost or utility or both are

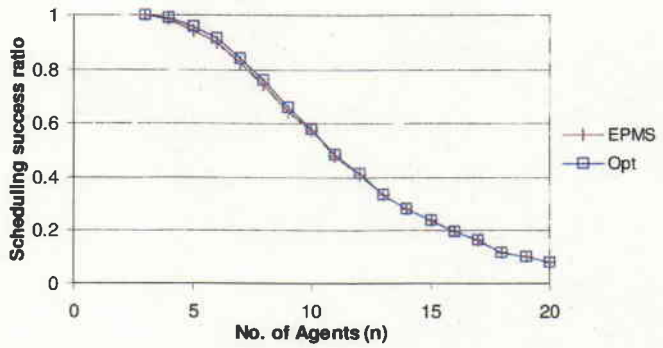


Figure 4.7: Scheduling success ratio in EPMS and the optimal solutions for varying number of participants.

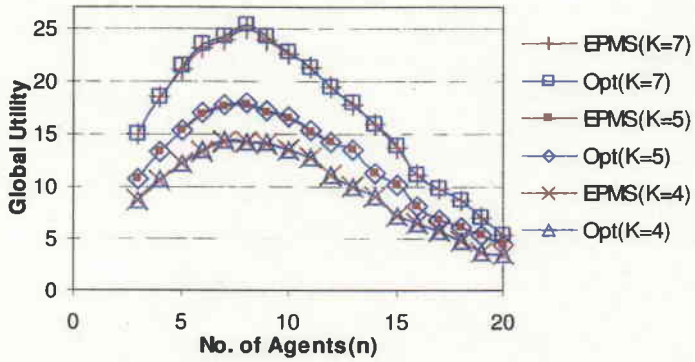
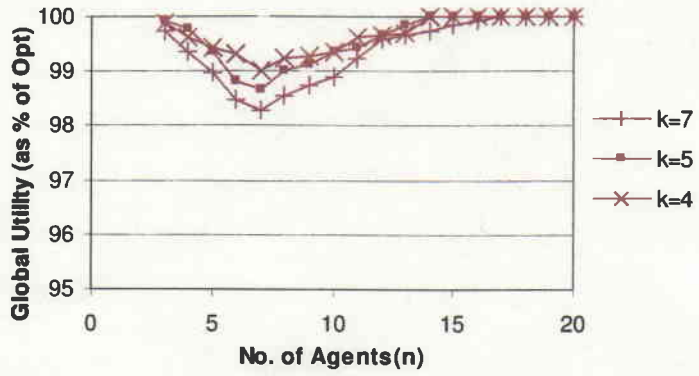


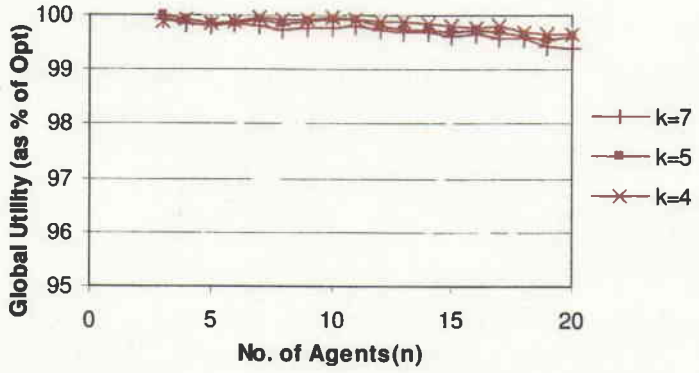
Figure 4.8: Relative global utility of EPMS and the optimal solution for varying number of participants.

included in the objective function), the shape of the curves in Fig. 4.8 will vary. However, the relative difference between the GU in EPMS and that in the optimal solution will remain the same.

To better understand the relative GU of EPMS with respect to the optimal solution, Fig. 4.9(a) shows the GU obtained in EPMS as the percentage of the GU obtained in the optimal solution with the over-constrained settings, i.e. the GU of the optimal solution is considered as 100%. The difference in the optimal solution is maximized around $n = 7$ for our settings. Also, the difference is larger when larger numbers of meetings (K) are scheduled concurrently. Figure 4.9(b) shows the GU obtained in EPMS with under-constrained settings, which is closer to the optimal solution than that with the over-constrained settings.



(a) Over-constrained.

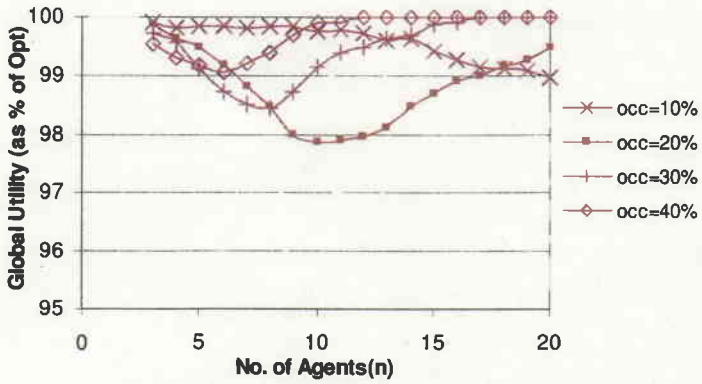


(b) Under-constrained.

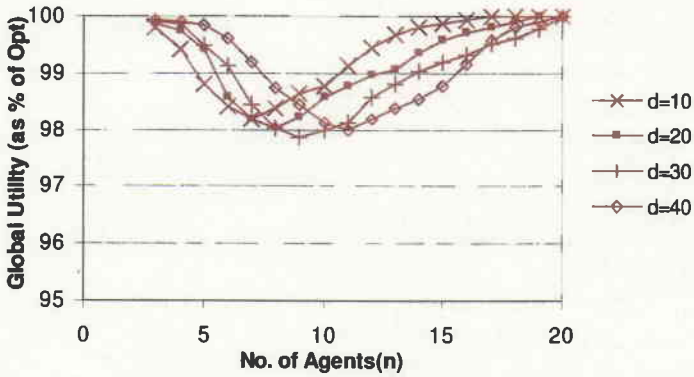
Figure 4.9: Global utility of EPMS as % of optimal solutions for different levels of constraints.

Figure 4.10(a) shows the GU obtained in EPMS as the percentage of the optimal solution, in scheduling seven meetings concurrently, for different values of average occupancy with previously fixed schedules (i.e. different levels of constraints) and Fig. 4.10(b) shows the same information but for different values of the number of candidate slots d. From Fig. 4.10(a), the worst performance of EPMS was found for the average occupancy level of 20% in our set of experiments. EPMS performs better (i.e. closes to the optimal) for lower occupancy levels and also for higher occupancy levels. Also, from Fig. 4.10(b), we see that EPMS performs better for smaller values of n with larger values of d, and for larger values of n with smaller values of d.

The graph in Fig. 4.11 shows the average GU obtained in EPMS for varying numbers of meetings scheduled concurrently. In this experiment, we considered that



(a) For different levels of occupancy.



(b) For different values of d.

Figure 4.10: Global utility of EPMS as % of optimal solutions for varying number of participants.

K meetings are scheduled in K slots (one meeting for each slot), any meeting can be scheduled in any of the slots from the same slot span, and the occupancy level of the participants was considered zero. From the graph, the average GU in EPMS is above 98% of the optimal solution, when small numbers of meetings are scheduled concurrently. However, it decreases slowly with the increase in the numbers of meetings scheduled concurrently.

Agent Communication Cost

The mobile agents in EPMS migrate to the agent server and after completing the job they return back to their hosts. On the other hand, stationary agents in distributed algorithms exchange remote messages. We measure the overhead incurred in both

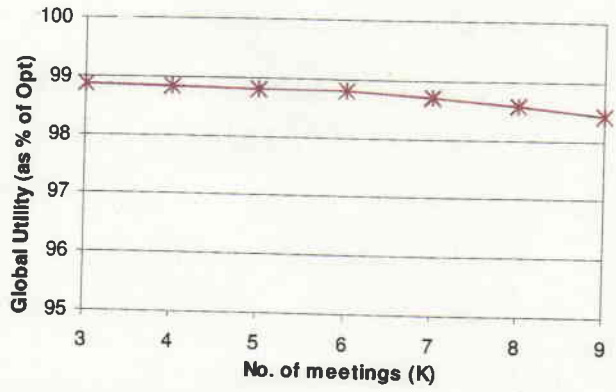


Figure 4.11: Global utility in EPMS as % of optimal solutions for varying number of meetings scheduled concurrently.

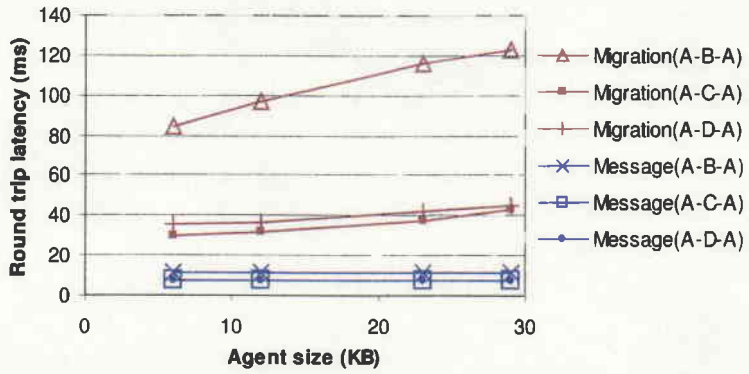
systems in a full-scale mobile agent platform. We used an open-source agent platform named Aglets [39, 33].

Experiments were carried out with four agent hosts (A, B, C and D) connected to local area networks with Ethernet cards. The hosts' specifications are shown in Table 4.4. We measured the latency and total traffic needed for agent migration and single messaging in the same environment. Their value varied in different runs due to other running processes and we took the average of 25 runs.

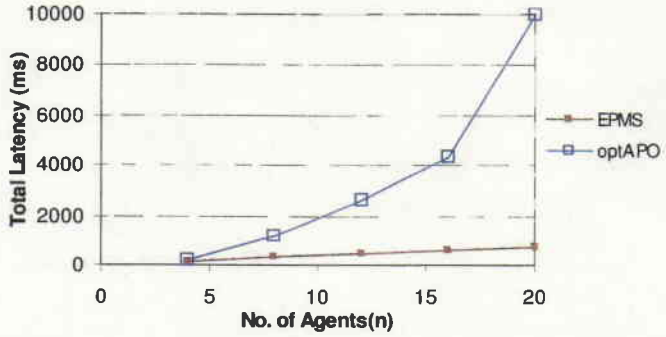
Table 4.4: Specification of the hosts' used in measuring latency and network traffic.

	Host A	Host B	Host C	Host D
Processor	1.4 GHz	600 MHz	1.2G Hz	1.8G Hz
RAM	768 MB	192 MB	512 MB	256 MB
NIC	100 Mbps	10 Mbps	100 Mbps	100 Mbps
OS	Windows XP	Windows XP	Windows XP	Windows XP

Figure 4.12(a) shows the average latency for agent migration of different sizes and that for inter-agent messages. The legend "Migration(A-B-A)"/"Message(A-B-A)" in the graph means the round-trip migration/messaging between host A and host B, starting from host A. Higher latency was incurred for both migration and messaging when host B, having low specifications, was involved. A single messaging latency between agents is small compared with agent migration latency. If the number of messages is small in stationary agent based systems, mobile agent based system are



(a) Single agent migration/single messaging round-trip latency for various agent size.

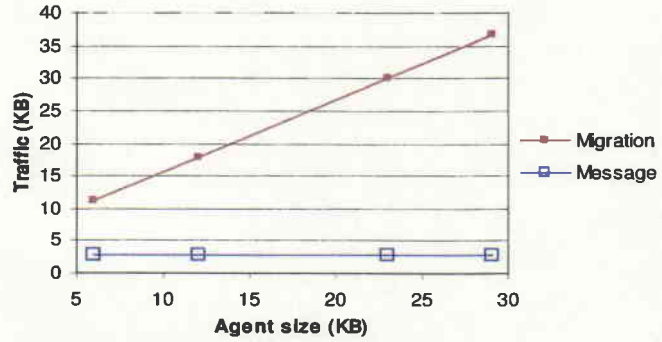


(b) Total latency for varying number of participants.

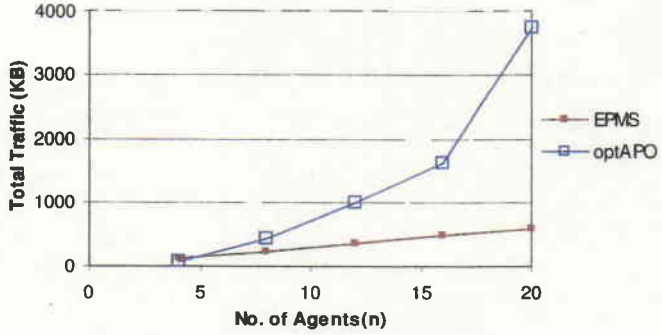
Figure 4.12: Agent migration latency and messaging latency in LAN.

more costly. But if the number of messages is large, the messaging cost becomes larger than total agent migration costs. The same type of advantages of using mobile agents has been shown by researchers [35]. The papers [26][46][44] show that stationary agent based distributed algorithms require a large number of messages. Figure 4.12(b) compares total agent migration latency in EPMS (agent size was 23KB) and total messaging latency in optAPO. Since the agent migration is one time round-trip for each agent, the total agent migration cost in EPMS is much less than that of the total messaging cost in distributed algorithms.

To measure total (TCP) traffic generated for agent migration and for inter-agent messaging between different hosts, we used a packet analyzer software [34]. Figure 4.13(a) shows the results for single agent migration/single messaging and Fig. 4.13(b) shows total traffic for varying number of participants n in EPMS and in optAPO. Here



(a) Single agent migration/single messaging traffic for various agent size.



(b) Total traffic for varying number of participants.

Figure 4.13: Generated TCP traffic for agent migration and inter-agent messaging in LAN.

also the mobile agent based solution (EPMS) generates less traffic as compared with stationary agent based solutions (optAPO) since stationary agents based solution requires a large number of messages.

Our measurement was on Aglet platform that uses Agent Transfer Protocol (ATP) [39]. The cost will be different in other agent platforms that use other protocols. However, since mobile agent migration is one time round-trip and stationary agent based systems require a large number of remote messages, in any platform the mobile agent based systems could cost less than the stationary agent based systems.

4.6 Discussion

In mobile agent-based applications, an untrustworthy host (or its user) may attack the visiting mobile agents by virtue of its superior control over them. So, the participating

agents should not visit any untrustworthy host. In EPMS, the participating agents need to migrate into the agent server where the common computational space is located. We assume that the agent server has been configured according to the trusted computing specifications [20][36] so that the users can trust it. Also, the users should make service agreement contracts with the service provider.

4.7 Conclusion

We have presented a mobile agent-based meeting scheduling scheme, EPMS, and evaluated it with respect to computational complexity, privacy loss, and global utility. The algorithm has a polynomial time complexity for scheduling multiple meetings concurrently. We measured its schedule privacy loss and preference privacy loss in the VPS privacy metric and Min privacy metric. We compared the privacy loss in EPMS with that in a well-known optimization algorithm, optAPO, and that in the centralized algorithm, which was shown to be efficient in the VPS privacy metric. The schedule privacy loss in the VPS privacy metric is the minimal in the centralized algorithm, while in the Min privacy metric, it is the minimal in EPMS. The preference privacy loss in EPMS is the minimal in both the VPS privacy metric and the Min privacy metric. The global utility in EPMS was found to be close to the optimal level for smaller numbers of meetings but decreases slowly for larger numbers of meetings. Thus, we can say that EPMS results in a good tradeoff among complexity, privacy, and global utility for solving the meeting scheduling problem.

Chapter 5

Reducing Privacy Loss using Agent Server

Traditional (non-cryptographic) algorithms can be classified as either centralized or distributed. A centralized algorithm is run by a single agent (participant) and results in a high privacy loss to the central agent, because all of the participants need to give away all their private information to the central agent. A traditional distributed algorithm is run by multiple agents (participants) and the participants do not have to disclose all their private information to the others. However, they need to disclose a fraction of their private information to others. Therefore, a traditional distributed algorithm does not offer complete privacy.

Cryptography based algorithms can solve the privacy problem by keeping the input data in an encrypted form. However, existing cryptography-based generalized solutions are too complex to be used practically. Researchers have developed a few problem-specific cryptography-based algorithms for solving specific MPCPs [11][51][64] based on the specific characteristics of the related specific problem. Even though the problem-specific solutions are more efficient than the generalized solution, the applicability of each of these cryptography-based algorithms is limited to only the specific problem for which it has been developed [51].

We present a new server-assisted privacy protection mechanism in solving many multi-party computation problems which does not use complex cryptography. First, we describe the basic idea of our privacy protection mechanism and analyze the traditional computing models to identify the requirements for realizing our basic idea. Then, we propose a problem solving approach, the security policies for the participating agents (including resource access policies and the policies for the computational result), and the security architecture of an agent server platform for enforcing those security policies. Rather than using cryptography-based protection mechanism, we use a server-assisted protection mechanism, which does not restrict our protection mechanism for only specific problems or specific algorithms. Thus, traditional (i.e.

non-cryptography-based) algorithms can be used to solve the problems.

5.1 Proposed Server-Assisted Privacy Protection Mechanism

The basic idea of our privacy protection mechanism is to solve the problem by sharing the private information among the participants in a controlled environment, but protecting the shared private information from being disclosed from the controlled environment.

Agents in traditional multi-agent systems are located at different hosts in a distributed architecture. It is very difficult to achieve uniform control over all of the participants in a distributed architecture and the private information, which is shared with other participants, cannot be protected from disclosure, because each of the participants is administered and controlled by individuals. In order to achieve the required control of our basic idea, (1) the architecture need to be a centralized architecture, (2) the participants need to migrate into the centralized architecture, (3) the participants need to bring all of their input data into the centralized architecture, (4) the architecture need to be capable of restricting the participants from disclosing the shared private information to others, and (5) secure computational result (containing no private information) should be sent to the users.

In our proposed server-assisted privacy protection mechanism, the participating agents are trapped into an agent server platform called iCOP (isolated Closed-door One-way Platform), which is the controlled environment for the participants. iCOP has the following characteristics.

Isolation: The agent server platform is isolated from the user hosts and the users retain no direct control over their agents in iCOP. The participating agents gather for interaction and negotiation at the iCOP instead of interacting at any of the user host platforms. This eliminates the necessity of accepting external agents into the user hosts where most of the user private data are stored and eliminates the risk on the user's local data store from external malicious agents.

Closed-door platform: iCOP is a closed-door platform from which the participants cannot communicate with the outside world. Agents can only communicate with the other agents that are brought into the platform. This characteristic of the

iCOP helps to protect malicious agents from leaking any kind of private data of other agents that are learnt in the problem solving process.

One-way platform: iCOP allows agents to only enter into the iCOP host platform with proper authorization but does not allow them to leave the platform. Accordingly the iCOP host is made as a one-way platform. On completion of their task, all the external agents, along with their data, are terminated at the iCOP host.

Following, we briefly describe the problem solution mechanism and then analyze its data disclosure channels to comprehend important suggestions leading to the protection technologies.

5.1.1 Problem Solving Mechanism

In our proposed mechanism for solving the multi-party computation problem, the participating mobile agents, along with their private input data, migrate into the iCOP (provided by a service provider) with proper authorization, share their private information within the iCOP through local messages amongst them, and carry out the computation within the platform (Fig. 5.1). Each user must be registered with the service provider and should sign a service contract. The service provider must have a privacy policy and the registered users should set their own privacy policies. The participating agents read the privacy policies from the users' settings and match with the policies of the service provider. If they match, the participating agents migrate into the iCOP host for conducting the desired computation. The trust relationship between a participating agent and the agent server is managed using digital signatures. A participating agent from a registered user is given authorization by checking the digital signature, which the agent carries with it.

Ideally, only the final result (e.g., in the MS problem, the date/time slot that all the agents have decided as the schedule of the meeting) for which they reach the agreement should be known to the respective users only, and any other information about the users should be protected from sneaking away by the agents.

Because of the closed-door and one-way nature of iCOP, the participants cannot send the computational result to the users by their own. However, without sending the computational result to the users, the system will be useless. So, the participants

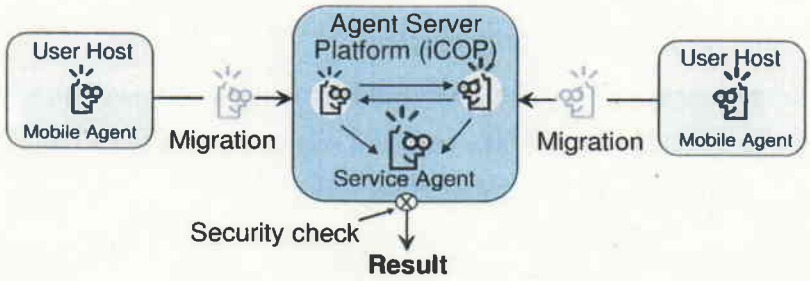


Figure 5.1: Proposed problem solving mechanism.

handover the computational result to a trusted stationary agent, called the service agent, which enforces the policies for the computational results (described in Section 5.1.3) for protecting them from leaking out hidden data through the computational result, and sends it to the users.

5.1.2 Problem Analysis

This section analyzes the probable data disclosure channels in the iCOP and finds the recommendations for realizing our basic idea. Data disclosure requires a communication channel (open or covert) between the sending and receiving entities and any communication channel between two entities requires using system resources. Thus, any communication between two entities can be protected if the sender can be protected from accessing the required system resources. However, if two entities should be allowed to communicate certain data (e.g., computational result) through a communication channel but should not be allowed to communicate other data (e.g., shared private information), then the communicated data must be checked and verified that they do not contain any encoded (hidden) data.

The participants should be granted access to necessary system resources to help them solve the problem and at the same time to protect them from leaking out the shared private information, all output channels accessible by them must be closed or otherwise controlled. Thus, *the security policy should not allow the participants access to any system resources that could be used to create open channels to transfer data to the outside world and should allow them access to the minimum system resources,*

which are mandatory for solving the problem.

The system must protect leaking any additional data through the result sending process. To send any additional data with the computational result, the data need to be encoded into it. Data can be encoded into the resulting object in many different ways (e.g., by adding directly, using steganography [3] etc.) and even the encoded data can be kept hidden by using cryptography. However, it is obvious that encoding any additional data into the resulting object produces a different object regardless of the content type of the object (text or binary) and encoding method of the additional data. Figure 5.2 shows a simple steganographic method for encoding data by changing some of the attribute values (marked with dotted ovals) of the original object [3]. It also shows a simple cryptographic method for hiding the encoded data.

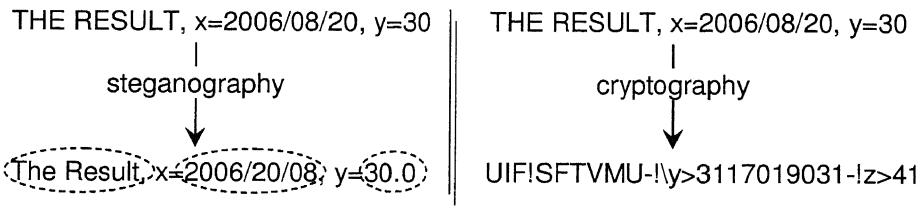


Figure 5.2: Encoding additional data into a text or binary object results in different objects.

In order to detect possible hidden data into the transferring computational result, which the security policy allows to be sent to the users, *the system security policy should ensure the availability of the original computational result (containing no hidden data) to be compared with the transferring computational result.*

In addition to protecting the private inputs of the participants, the computational result created by the participants should also be protected from others, including the service agent. So, *it should be encrypted with a group key of the participants* [67] so that a non-participant cannot decrypt it.

Besides open channels, covert channels can be created through the shared resources or objects between the participants and the outside world. The sender changes the attribute values of the shared resources or objects to encode/signal the data and the receiver reads the changes in the attribute values of the shared resources to interpret the encoded/signaled data. Such communication channels are generally known

as covert channels [19]. Figure 5.3 shows a block diagram of the covert channel.

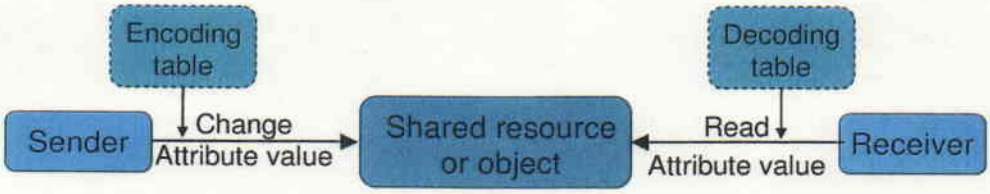


Figure 5.3: Block diagram of covert channel.

A covert channel can be classified either Storage Channel or Timing Channel. Various examples of covert channels can be found in paper [19]. In a storage channel, the sending process alters a particular data item, and the receiving process detects and interprets the altered data to receive information. For example, the sender can signal bits by locking and unlocking a file and the receiver can interpret the bits by reading the locked status of that file. Both the sender and the receiver need to access the lock status of the same file. In a timing channel the sending process modulates the amount of time it takes for the receiving process to detect a change in an attribute value or to perform a task, and the receiving process interprets this delay or lack of delay as information. A covert channel is noisy if the corresponding shared object is available to other processes as well as to the sender and receiver and its attribute values are modified by more than one processes; it is noiseless if the corresponding shared object is available only to the sender and the receiver and its attribute values are modified by only the sender [19].

We can protect using covert channels with the outside world by a potential senders by not granting access to the resources to the potential senders (i.e. not sharing) and by protecting them from changing the attribute values of the shared resources or objects. In addition to closing/controlling all possible open channels with the outside world, *the system must detect and handle all of the possible covert channels* through which data can be transferred in a covert/hidden manner.

System security policies define the set of rules which the participants should follow. However, without any policy enforcement mechanism, the policies do not guarantee information security/privacy. Thus, *the system security architecture must have a policy enforcement mechanism to enforce the defined policies.*

5.1.3 Security Policies

The security policy of our agent server platform consists of the resource access policies and the policies for the computational result.

Resource Access Policies

Some system resources, such as the memory and CPU, are mandatory for every program for their normal operations. So, our resource access policies allow the participating agents access to the mandatory resources (such as OS, CPU, and memory) at the server. The participating agents cannot create any communication channel with an outside entity by using only the mandatory resources, which are granted to them. In order to solve the problem, they need to share their private information. So, our security policies allow them to exchange local messages and share their private information from within the iCOP. However, they are not granted access to any other system resources, such as files and network sockets, which can be used to create communication channel with outside the iCOP to transfer the shared private information directly.

For carrying out various tasks, including the coordination among the participating agents, sending the computational result to users, and terminating the participating agents (after the computation is over), our resource access policies grant the service agent all-permission.

Policies for the Computational Result

In order to protect the participating agents from leaking the shared private information through the result sending process, the service agent enforces two policies before sending the result R to the users: (1) each participating agent A_i must pass the computational result R_i to the service agent and (2) the computational results passed by the participating agents must be identical, i.e. $R_i = R_j \ \forall i, j \in \{1..n\}$, where n is the number of participants. The first policy helps in making the original computational result available to be compared with any suspicious computational result and the second policy helps in detecting any encoded private information in

the computational result. If at least one of the participating agents A_i does not include any private information into its computational result R_i i.e., creates the actual computational result $(R_i =)R$ then any encoded data into the computational results R_j created by any other agent A_j can be detected by comparing R_j with R_i. Thus, the enforcement of the above two policies can protect the participating agents from leaking the private information through the computational result, provided that at least one of the participating agents does not include any private information into its computational result.

In order to meet the first condition, each participating agent A_i must actively participate in the computation and get its copy of the result R_i by using any simple algorithm. To meet the second condition, i.e. to make identical computational results from their components, we propose to use a pre-defined format in creating the computational results from their components. Without using a pre-defined format, individual agent may make the same result in different formats (and thus hide private information into it) making them non-identical.

Result Format

The computational result consists of a set of variables and constants. The result format defines the rules of constructing a single message with its components' name-value pairs so that the result created by individual agent match with each other. Each of its component value has a data type. We broadly classify the data types as (1) integers (2) real numbers (3) date (4) string, and (5) boolean and assume that the variable names are string. The format should define at least following characteristics (where applicable).

1. Component format

   - Data type of each component (integer, string, date etc.)
   - Character case of string type
   - Number of digits after decimal point of real numbers
   - Date format ("yyyy/mm/dd", "mm/dd/yyyy" etc.)

- Boolean value (T/F or True/False)

2. Appearance order of the component_name-value pairs.
3. Separation character between component_name-value pairs.

Suppose, the result consists of four components- one constant and three variables x, y and z. Figure 5.4 shows a computational result in six different formats. Even if they are semantically equivalent, they are not identical. The differences in the objects in lines 2-6 with the object in line 1 are marked with dotted ovals.

THE RESULT, x=2006/08/20, y=30
The Result, x=2006/08/20, y=30
THE RESULT, x=2006/08/20, y=30
THE RESULT, x=2006/08/20, y=30.0
THE RESULT, x=2006/20/08, y=30
THE RESULT, y=30, x=2006/08/20

Figure 5.4: Different formats may produce non-identical computational result from its components.

By using a pre-defined format, say (1) component format- {constant}<string> <upper-case>, x=<real> <two digits> y=<integer> and z=<date> <yyyy/mm/dd>, (2) appearance order- {constant}, z, y, x, and (3) separation characters- a comma followed by a space ", ", the resulting object R_i created by the individual agents A_i will be identical to the object in the first line in Fig. 5.4.

The rules mentioned here are not fixed for every context, but it is necessary to define those characteristics of the components to create a guideline for the participating agents to create identical results (messages) that are to be matched with each other by the service agent. In general, fixing the string case or a specific date format or the appearance order of the components does not affect the semantic much. Allowing different formats to represent the result creates the scope of hiding information into it and enforcing them to a fixed format prevents data hiding into it.

In order to keep the computational result secret from the service agent and any other unauthorized parties, we also propose that the participating agents should encrypt the computational result using a group private key so that a non-participant cannot know the computational result.

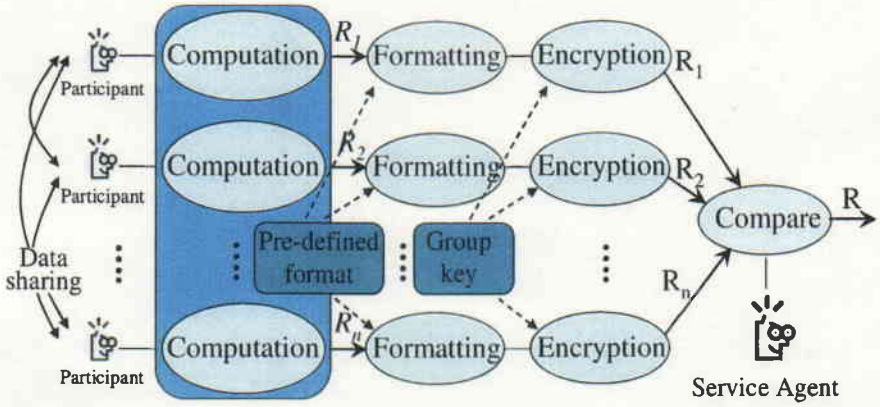


Figure 5.5: Process of creating and sending computational result.

The process of creating and sending the computational result R in our mechanism has been illustrated Fig. 5.5 The figure shows that each of the participating agents shares its private information, actively participates in the computation to get its copy of the computational result, uses a pre-defined format to create identical computational result, encrypt the computational result with the group key, and passes its copy of the encrypted result to the service agent, which compares them with each other for equality and sends to the users.

5.1.4 Security Architecture

iCOP architecture consists of two basic units: a) Management or control unit and b) Computational unit. Figure 5.6 shows a simple conceptual diagram of iCOP architecture. The computational unit, consisting of the participating agents, performs basic computations. The input data, which it needs to solve the problem, come along with the agents through the input channel. The management unit oversees the operations of the computational unit, monitors and controls the resources, which the computational unit may use to perform its computation. The management unit exercises access control over the computational input channel and output channel.

In addition to defining the security policies, a protection mechanism must have a policy enforcement technology. Accordingly, the proposed iCOP security architecture has a privacy manager for enforcing the resource access policies and a service agent for enforcing the policies for the computational result (Fig. 5.6).

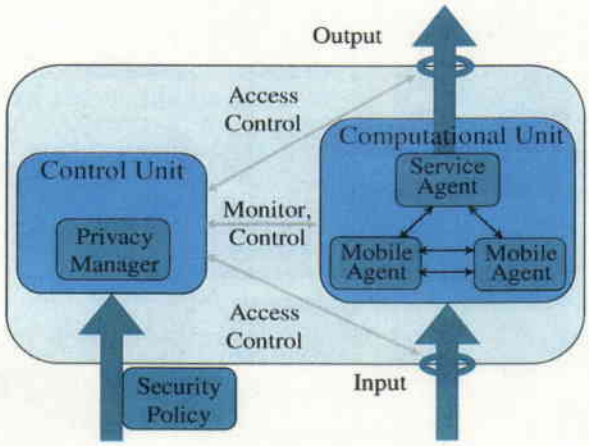


Figure 5.6: iCOP security architecture.

The privacy manager is a reference monitor that monitors and controls the access to the system resources by the agents in iCOP. It restricts the participating agents from accessing the system resources without which they cannot communicate with the outside world to disclose the shared private information. When any direct or indirect request to access any of the system resources is made by an agent in iCOP, the privacy manager holds the request, checks if any of the participating agents are involved in the access request, and grants or denies the access based on the system security policies. To make sure that the participating agents are not allowed any unintended access (by virtue of high level permission) to the system resources, the privacy manager explicitly checks each permission. For any access request to the optional system resources, which are not allowed by the resource access policies, the privacy manager inspects the stack frame execution context (the list of classes in the system class stack) corresponding to the current series of method calls and throws exceptions (i.e. deny the request) if it finds any external agent in the stack having a codebase other than the server host. Thus, the privacy manager enforces the resource access policies and makes the platform one-way and closed-door.

To enforce the policies for the computational result, we assume that the service agent is a trusted agent. The service agent gets the computational result from the participants and sends it to the users, provided that the computational result conforms to the defined security policies. Section 5.2.3 describes how the enforcement of

those two policies can restrict the participating agents from leaking out the shared private information through the computational result. Finally, in order to destroy the shared data, which the participating agents acquire in the problem solving process, the service agent terminates the participating agents along with their data at the agent platform.

5.1.5 Service Protocol

A registered user can initiate the service by sending a request to the service agent. The initiator agent must give all of the initial parameters to the service agent before it migrates into the iCOP host. The service agent invites other participants to join the computation and provides the initial parameters to them including list of participants, problem description such as name of the problem, list of variables to be assigned, the domain of the variable values, the domain of personal valuation of certain variables etc. The parameters that are sent to the service agent before the participating agents migrate into iCOP and before they share their private information among them are treated as the initial parameters. The initial parameters cannot contain other agents' private data because those parameters are sent before private information is shared. Thus, it is safe to send them out of iCOP host with the invitation. Upon getting the invitation, all participating agents collect related necessary data based on the supplied initial parameters for the computation and migrate into the iCOP host. The participating agents migrate into iCOP with their input data. This migration is controlled and checking digital signatures, which the participating agents carry with them, so that only the registered users can send their participating agents. The participating agents carry out the computation by sharing their personal data through local messages.

Each of the participating agents create the computational result in the process illustrated in Fig. 5.5 and hands over to the service agent, which compares the results for equality, send the result to the users and terminates (disposes) the participating agents. Figure 5.7 illustrates an example of a simple service protocol diagram.

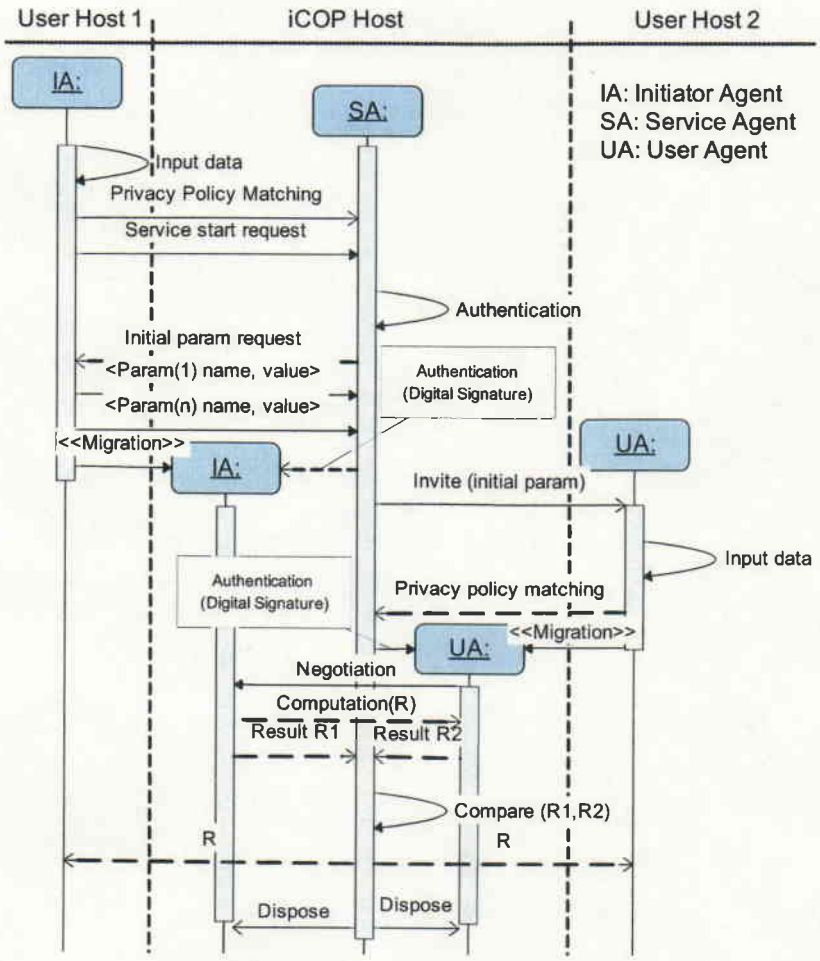


Figure 5.7: Service protocol sequence diagram.

5.1.6 Reliability and Scalability Issues

A single server is subject to a single point of failure. In case of using a single server, if the server fails or somehow compromised, the whole system fails. In order to handle the fault tolerance, increase the reliability and availability of the server, and achieve better scalability, we need multiple iCOP servers. These servers make up the iCOP domain. After checking digital signatures of incoming agents, the authentication server dispatches all of the participating agents of the same application to one of the several computation servers (Fig. 5.8). Each of the computation server of the iCOP domain must be made closed-door and one-way. By distributing different applications into different computation servers in iCOP domain, we can achieve scalability and

load balancing. If one the server systems fails, only the applications in that server will fail, the rest of the servers can continue their computation. All of the applications will not be affected by the failed system. Thus, we can achieve application-wise fault tolerance. The use of replication server or backup server also provide fault tolerance of the whole system.

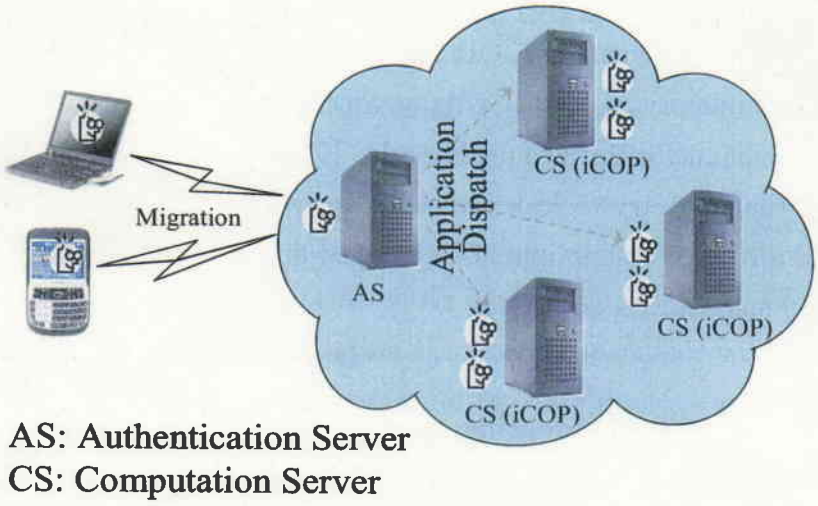


Figure 5.8: iCOP domain with multiple agent servers.

If different agents of the same application could be distributed into different computation servers, we could achieve in-application fault tolerance for which if one of the servers fails, the rest of the applications could continue computation. However, since the participating agents of the same application need to communicate among them, in-application fault tolerance needs inter computation server communication, i.e. resource permission for the participating agents. As previously explained, allowing resource permission (even within iCOP domain) would create possibility of covert channel between iCOP and other programs at the server. So, in order to avoid such covert channels, we do not distribute the agents of the same application into different computation servers.

5.2 Security and Privacy Analysis of the Proposed Mechanism

It is a common assumption in multi-party computation domains that a secure communication channel between any two hosts exists. Also, we assume that standard language level safety and operating system level safety are maintained in the server. So, because of the limited resource permissions, the participating agents cannot attack each other or the platform. This section shows through intuitive analysis how the enforcement of the security policies protects from information leakage.

The participating agents in iCOP are not granted access to any system resources (other than mandatory resources) without which the participating agents cannot create any open channel with the outside world. However, there may be covert channels being used that may try to leak the shared private information. All of the covert channels in a trusted system must be identified and handled [19]. Thus, we perform a covert channel analysis on iCOP.

5.2.1 Covert Channel Identification

We use the shared resource matrix method [37] to identify potential covert channels in iCOP. With this method, if an attribute of a shared resource is found that can be modified and referenced by two different processes, which are not allowed to communicate through legal channels, then potential covert channels exist through that resource.

Every process at the agent server, including those of the participating agents at iCOP, uses mandatory system resources (Fig. 5.9). A participating agent in iCOP may try to modify the attribute values of those resources and the external processes may be able to refer to those attribute values. So, potential covert channels between the participating agents and other programs in the system may exist through the mandatory system resources.

The computational result from the participating agents is sent to the outside receivers (i.e. users) by the service agent, i.e. the resulting object is shared between the participating agents and the users by transferring the object itself. Thus, potential covert channels between the participating agents and the users may exist through the

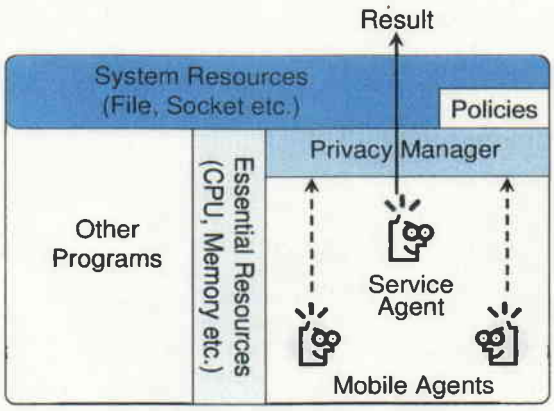


Figure 5.9: Shared server resources.

computational result.

Our security policies do not allow any other objects or resources to be shared between the participants and any other entity outside the iCOP. So, there can be no covert channels between the participating agents and the outside entity through any other resources/objects. Table 5.1 summarizes the data disclosure channels between the participating agents in the iCOP and the outside world.

Table 5.1: Summary of data disclosure channels in iCOP.

Chan. Type	**Through**	**Exist?**
Open	Any resource or object	No
Covert	1. Mandatory system resources	May exist
	2. Computational result	May exist
	3. Optional system resources	No

5.2.2 Potential Data Leakage Mechanisms

A potential sender (participating agent) may try to leak out the acquired private data though communication channels, which it can access. There are no open channel accessible to the participating agents. However, two types of covert channels may exist in iCOP: through the mandatory system resources and through the computational result.

A participating agent may create covert channels through mandatory system resources by implicitly changing their attribute values. For example, the sender may signal a 0 through relinquishing CPU by going into sleeping mode and may signal a 1 by running CPU extensive process; the receiver measures how quickly it gets CPU and detects 0 if delay is low and detects 1 if the delay is high. Also, the sender may exhaust memory to signal, say 1, and releases memory to signal, say 0. The receiver need to try to allocate memory and depending upon success or failure, it can detect signal 0 or 1.

The computational result created by the participating agents is transferred to the users. Thus data leakage though any covert channel via the computational result requires to encode (and hide) the encoded data into it. Hiding data into an object is generally known as steganography [3] and requires some kind of modulation on the original object. In our context, the original computational result refers to the actual computational result which does not contain any hidden data.

The modulation technique uses some kind of protocol to encode information and the receiver need to perform related demodulation to interpret the encoded data. Different kinds of modulation are possible. Following are some examples.

Adding text: The sender adds additional characters like white space, punctuation marks or decimal point in numerical values etc. without changing the semantic of the text. Table 5.2 shows an example of simple protocol and Fig. 5.10 shows an example of hiding data using that protocol. If the first line of Fig. 5.10 is the original object, then with the protocol shown in Table 5.2, the second line of Fig. 5.10 contains hidden data "010000011110". But, semantically both lines are equivalent.

Table 5.2: Example of a simple steganography protocol by adding characters

Signal	Means
Additional 1 white space	00
Additional 2 white space	01
Additional 1 digit with numerical value	10
Additional 2 digit with numerical value	11

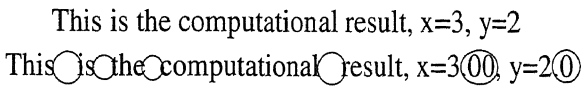


Figure 5.10: Process of encoding data into computational result by adding extra characters.

Arranging components: An object may possess the same semantic even after re-arranging its distinguishable components. For example, a date may be represented with different formats. A simple steganography protocol can be built using different formats as shown in table 5.3. With the protocol shown in Table 5.3, to send "101" the sender must send a date value in the "yyyy/dd/mm" format. Similarly, by using different sequence of components of the computational result, a number of bits can be transferred covertly.

Table 5.3: Example of a simple steganography protocol by arranging components

Format	Means	Format	Means
dd/mm/yyyy	000	mm/yyyy/dd	011
dd/yyyy/mm	001	yyyy/mm/dd	100
mm/dd/yyyy	010	yyyy/dd/mm	101

Changing case: Data can be encoded by changing the case of certain alphabets (e.g., start of each sentence, first alphabet of each word or an alphabet of any position) of the text. With the protocol shown in table 5.4, the text "This is the Computational Result" contains hidden data "10011".

Table 5.4: Example of a simple steganography protocols by changing case of alphabets

Format	Means
Capital letter	1
Small letter	0

Above, we have shown few examples of data encoding/hiding techniques. However, there can be many other techniques of encoding data into an object and even the encoded data can be kept hidden by using different techniques e.g., cryptography.

However, it is obvious that encoding any additional data into the resulting object produces a different object regardless of the content type of the object (text or binary) and encoding method of the additional data.

5.2.3 Covert Channel Handling

Two types of covert channel may exist in iCOP: thought the mandatory system resources and though the computational result. In this section we describe how these covert channels are handled in the iCOP.

The covert channels through the mandatory system resources are very noisy i.e. their bandwidth is very low, because all the processes in the system use the mandatory system resources. This type of covert channel is common in every system, because in every system the mandatory system resources are shared among all system processes. Thus, the underlying operating system handles these covert channels using various techniques (e.g. memory partitioning, CPU scheduling etc.) to eliminate them or to reduce their bandwidths to very low values making them ineffective [56][19]. So, we do not take any special measures for them. As an additional measure, their bandwidths can be further reduced by introducing additional delays and noise deliberately into those channels (e.g., using random allocation algorithms; introducing extraneous processes that modify covert channel variables in random patterns) [19].

The enforcement of the two specific policies (conditions) set forth for the computational results can protect the participants from using covert channels through the computational result. A covert channel through the computational result is possible only when the result is sent to the users by the service agent. Let us suppose that the actual computational result is not equal to some private data of some participant. Let us suppose that an actual computational result containing no private information is R. Additional data can be encoded into the computational result using many different techniques [3][31] and the encoded data can even be kept hidden by using cryptography (Fig. 5.11). However, it must be noted that encoding any additional data into the resulting object produces a different object, regardless of the content type of the object (text or binary) and encoding method of the additional data (Fig. 5.2, 5.4 and 5.11). In other words, if a suspicious object is identical with the original

one, which contains no hidden data, then the suspicious object cannot contain any hidden data. Thus when a participating agent A_i encodes some private information with its computational result R_i will not be identical to the actual computational result R.

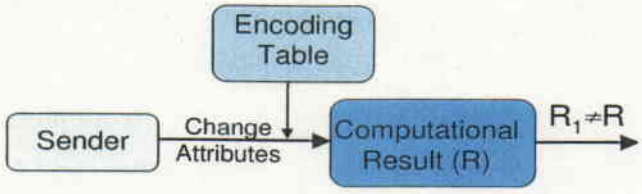


Figure 5.11: Encoding data into the computational result by changing attribute values creates different object.

We categorized the possible maliciousness, associated relationships among the computational results passed by the participants, and the corresponding inferences for the enforced policies into five cases (Fig. 5.12).

Case	Malicious	Non-malicious	Created Results	Policy met? (data sent?)	Privacy loss
1	A_1	$A_2..A_n$	R_1=null, R_2=...R_n=R	No	No
2	A_1	$A_2..A_n$	$R_1 \neq R_2$=...R_n=R	No	No
3	$A_2..A_n$	A_1	R_1=R $\neq R_2$=...R_n	No	No
4	None	$A_1..A_n$	R_1=R_2=...R_n=R	Yes	No
5	$A_1..A_n$	None	R_1=R_2=...$R_n \neq$R	Yes	Mutual

No agent cares about its own privacy

At least one agent cares about its own privacy

Figure 5.12: Possible cases of the relations among the computational results from different participants.

In Case 1 in Fig. 5.12, one of the participants, say A_1, has not provided any computational result to the service agent. So, the policies are not met, and the result is not sent to the users, i.e. no scope for creating a covert channel. In Case 2, one of the participants, say A_1, is malicious; it has encoded some private information into its computational result. So, the computational results will not be identical, the policies

are not met, and the result is not sent to the users, i.e. no scope for creating a covert channel. In Case 3, most of the participants are malicious and they have encoded private information with their results, which are not identical to the non-malicious participant. Thus, the policies are not met, the computational result is not sent to the users, and there is no scope for creating a covert channel. In Case 4, none of the participants are malicious, they have created identical computational results (that are equal to the actual computational result) containing no hidden information, i.e. no covert channel has been created, the policies are met, and the result is sent to the users. Finally, in Case 5, all of the participants have been considered malicious and each of them has encoded private information into their results. If they make identical results, the result is sent to the users and it causes a mutual privacy loss.

From the above analysis, we see that (mutual) privacy loss is possible when all of the participants are malicious (i.e., Case 5). If at least one of the participants cares about its own privacy and wants to protect its own private information from being leaked out, all it needs is to be not malicious and make its own result identical to the actual computational result, which eliminates the possibility of a Case 5 and there remains no scope for privacy loss. Thus, the enforcement of the policies set forth for the computational results can prevent encoding hidden data into the computational result for leakage.

5.2.4 Other Methods of Leaking Data

During our investigation, we found an example of how to make it possible to signal hidden data to the users in the result sending process without encoding the data into the computational result. We have devised one such method, which we call the result biasing method. If there are a number of possible solutions to a given problem, a protocol can be created that maps one bit stream with each of the solutions numbers (Table 5.5). To send one of the bit streams, which are defined in the protocol, an agent needs to manipulate its own inputs in the algorithm so that the final result leads to the respective solution number in the protocol. For example, Fig. 5.13 shows four possible solutions and their flow. After observing the given value of a1, Agent B may give a value for b1 so that the solution leads to either S3 or S4. Then, after

observing the given value of a2, Agent B may give a value for b2 so that the solution leads to S3. Thus, Agent B can lead to the solution number S3 when it needs to send a "10" (according to the protocol in Table 5.5) to its users. Agent A is not aware of such input manipulation. , When the user receives solution no. 3 as the computational result, she can recognize that her agent has sent a "10". We see that hidden data can be sent through this method in the result sending process without encoding it into the computational result.

Table 5.5: Simple example protocol for result biasing method.

Soln. no.	Mapped with	Soln. no.	Mapped with
S1	00	S2	01
S3	10	S4	11

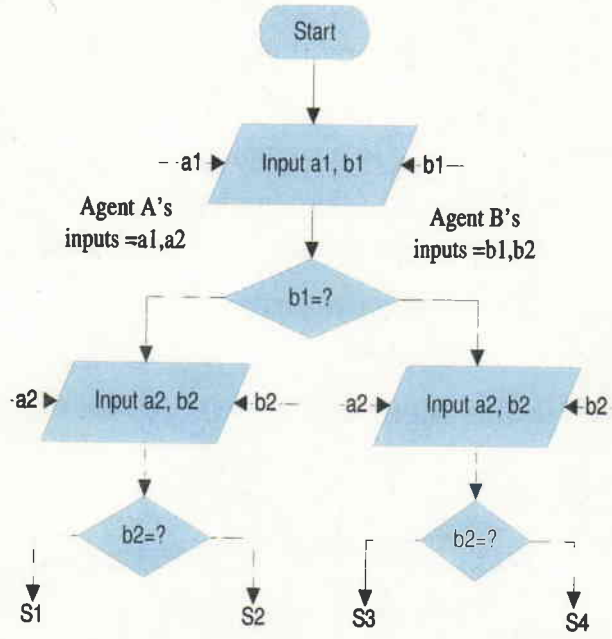


Figure 5.13: An agent can lead to one specific solution by manipulating own inputs.

Leaking data through the result sending process without encoding it into the resulting object requires the existence of multiple solutions of the problem, they are priory known (for creating the decoding table) and an agent can successfully bias all the other agents towards a specific solution.

5.2.5 Complete Privacy Protection

There are distributed algorithms as well as centralized algorithms for some problems (e.g. the meeting scheduling problem). However, there are no distributed algorithms for some other types of problems (e.g. the vector dominance problem). These problems are solved with centralized algorithms. In the iCOP, even though the participants are protected from encoding the shared private information into the computational result and disclosing it to the users, they can not be protected from manipulating their own inputs during execution of a distributed algorithm and thus signaling hidden data to the users as described in Section 5.2.4. However, the manipulation of the inputs can be protected and complete privacy can be achieved if the inputs are exchanged first with a commitment protocol [49] and then centralized algorithms are used by each of the participants to solve the problem. Simple exchange of inputs also allows input manipulation and thus result biasing. For example, in a simple input exchange (Fig. 5.14 after getting values (a1,a2) from Agent A, Agent B can calculate its required value of b1 and b2 for leading the solution number to the desired one and then give those values to Agent A. In a commitment protocol (Fig. 5.15, the values need to be committed before actual values are exchanged. In the verification phase, any mismatch between the committed values and the given values can be detected. Thus, after committing values, if an agent gives different values than the committed ones by observing the other parties inputs, the other party can detect it.

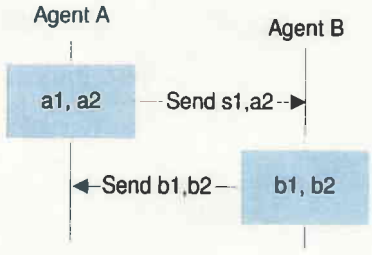


Figure 5.14: Simple input exchange.

Our system requires that all of the participants must give identical result to the service agent. So, if a centralized algorithm is to be used in iCOP, in order to meet the conditions for the computational result, each of the participants should execute a centralized algorithm individually with the committed inputs. Thus, they can be

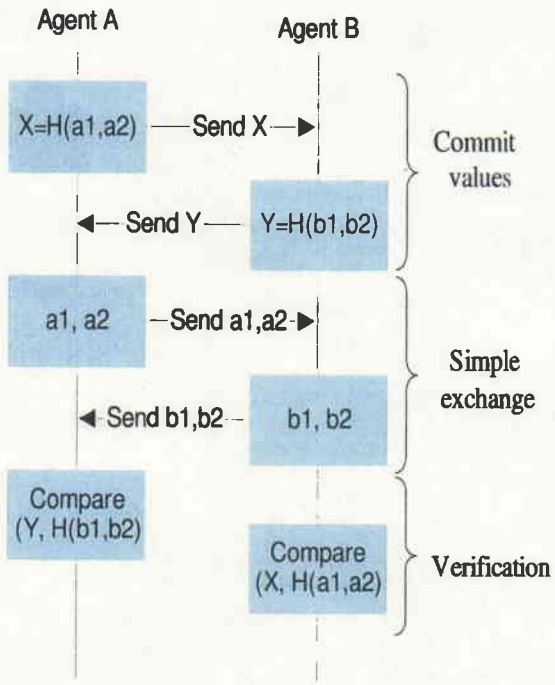


Figure 5.15: The commitment protocol for exchanging inputs.

protected from manipulating their inputs for biasing biasing the result to signal any data to the users. However, executing multiple centralized algorithm at a the server will require much higher computational time than executing a distributed algorithm. So, complete privacy protection is achieved at a cost of higher computational time.

Figure 5.16 shows a summary of the data disclosure channels in iCOP and their protection schemes. All of the open channels are protected by the privacy manager. The possibility of creating covert channels, which requires shared resources, are protected by not sharing objects/resources, except the computational result, between the participating agents and the outside world. Enforcing the policies set forth for the computational result protects most of the scope of creating covert channels through the computational result. In addition to enforcing these policies, the participants can ensure complete privacy by committing inputs and solving the problem themselves using centralized algorithms.

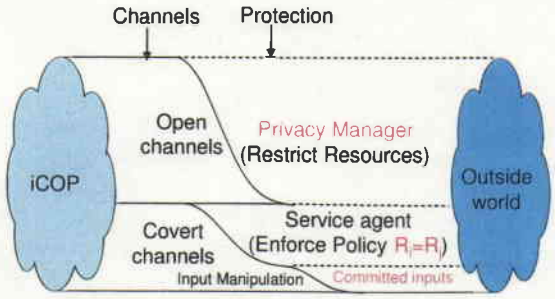


Figure 5.16: Summary of data disclosure channels in iCOP and their protection schemes.

5.2.6 Implementation Issues

The Java architecture has a reference monitor, which can monitor and control the use of system resources [23]. But, the Java security manager enabled JVM itself is not a one-way closed-door platform. Thus, we use the Java architecture and customize the Java security manager to make the platform one-way and closed-door. Even though Java technology can provide security to end user system against untrusted code (e.g., applet from the Internet) by putting them into sandbox and protecting them from security sensitive activities on local system, it allows downloaded code in the sandbox to connect back to the originator i.e., open channels between the sandbox of end user host and the originator host of the agent can be created. So, if an agent can get some information (due to sloppy security policy) from local system, it can send the information to its originator. Even the server can be made secured from mobile agents by activating the Java security manager with proper policy, the agents of multi-party computation exchange their private information in their problem solving process, which is unavoidable. Thus, the agents get sensitive information at the first place (due to the nature of the multi-party collaborative problem) without compromising the security manager and can send the acquired information to their originator hosts.

Customizing the Java security manager to close all channels, which may be used by the mobile agents, can prevent them from leaking out any information. However, without sending the computational result to the users, the system becomes useless. It needs a mechanism to send the computational result to the users. For this, iCOP uses a trusted service agent, which is assumed to be not malicious. To ensure that

the mobile agents don't send hidden information through the result sending process, the service agent enforces the security policies defined for the computational result.

We developed a prototype server for multi-party computation based on the security policies and security architecture described in Sections 5.1.3 and 5.1.4, respectively. We used the Aglet environment (Ver. 2.0.2) [39][33], which is a Java-based open-source agent development kit. The original Aglet platform is not a closed-door one-way platform. We customized the Aglet security manager (which in turn is a customized Java security manager) to make the iCOP privacy manager. We also modified the parts of the Aglet platform that do not conform to our security policies.

The default Aglet environment has a retract facility that allows the users to retract their agents from an Aglet server. We removed that facility in the iCOP server, by modifying its source code, so that the users cannot retract their agents from the iCOP. To allow exchange of local messages, the participating agents were granted Aglet defined message permission. However, with experiments, we found that the Aglet defined message permission includes some Java defined permissions e.g., network socket permission. To make sure that the participating agents are not allowed any unintended permissions (by virtue of implied Java defined permission caused by any Aglet defined permission) on the system resources, the privacy manager explicitly checks each permission on system resources. For any access request to the optional system resources, the privacy manager inspects the stack frame execution context (the list of classes in the system class stack) corresponding to the current series of method calls and throws exceptions (i.e., deny the request) if it finds any external class (mobile agent) in the stack having a codebase other than the server host. Thus the privacy manager restricts the external agents (i.e. participating agents) from using any of the system resources other than the mandatory resources, and makes the platform one-way and closed-door.

In our prototype, the service agent was made a simple stationary agent. After it gets the computational result from any of the participating agents, it waits for a threshold amount of time to get the results from all other participating agents. Then it checks if all of the present participating agents have given the results and if the result from them are identical. If both the conditions are met, it disposes the

participating agents and sends the result to the users.

5.3 Experimental Evaluation and Analysis

5.3.1 Effectiveness of the Privacy Manager

For evaluating the effectiveness of the privacy manager in protecting mobile agents from disclosing the shared private data to the outside world, we created several malicious mobile agents with the same functionalities of the hostile applets reported by the Java community [38]. Besides, we created several other mobile agents to test the capability of the privacy manager in protecting them from performing other forbidden operations in the iCOP. Then, we sent those mobile agents into the iCOP server and observed whether they were successful in performing their malicious tasks.

Table 5.6 lists the mobile agents of our experiments, the operations that they tried to carry out in order to leak the personal information, and the protection result of the privacy manager. Following are the short descriptions of the agents.

Besides direct message communication, a mobile agent may send sensitive data to external world by e-mail. The *mailer agent* is an agent capable of sending e-mail to any address. The maliciousness of this agent we considered was to send some information by e-mail.

The *re-director agent* tried to make connection to any host by redirecting the connection with the originator. The *URL opener* agent also tried to open an URL connection with any host in the network.

The *thread killer* is a type of agent which tried to kill a thread in the system. If it becomes successful, it can kill the threads of all other agents including the service agent.

The *mutator agent* is capable of modifying an existing class file. A mutator agent may modify the class file of the service agent and modify its functionality to work for the mutator agent.

The *messenger agent* can send messages to its originator. This agent was given a host address and a directory name of the host. It listed the contents of the given directory and tried to send back the listing to the originator.

Table 5.6: List of malicious agents, their operations and the action taken by the privacy manager.

Agent	Operations	Action by the Privacy Manager
Mailer	Send e-mail to the origin host	Blocked (Lack of SocketPermission)
Re-director	Re-direct the connection to a different URL	Blocked (Lack of SocketPermission)
URL opener	Open a new URL	Blocked (Lack of SocketPermission)
ID, Password sender	Asks for the ID and password and sends them to the origin host	Blocked (Lack of SocketPermission)
Thread killer	Kill all threads except itself	Blocked (Lack of modifyThread per.)
Mutator	Modify class files located at local host	Blocked (Lack of FilePermission)
Messenger	Send messages to a remote agent	Blocked (Lack of SocketPermission)
RMI client	Open connections to an RMI server	Blocked (Lack of SocketPermission)
Agent creator	Create Agents from the origin host	Blocked (Lack of SocketPermission)
Cloner	Create clones of itself	Blocked (Lack of Clone Permission)
Dispatcher	Send agents to the origin host	Blocked (Lack of SocketPermission)
Migrator	Migrates to the origin host	Blocked (Lack of SocketPermission)
Retractor	Retract agents from the origin host	Blocked (Lack of SocketPermission)

The *RMI client* mobile agent is an RMI client that can send message to a remote RMI server. A malicious mobile agent may try to send unauthorized data to a remote RMI server by invoking RMI methods.

The *agent creator* agent can create agents from the origin host. Thus, an agent creator may send signal to the origin by creating/not creating agents from the origin host at some intervals indicating "0" or "1".

The *cloner* agent may create its clone and leave its copy at the server. Thus it may avoid the termination by the service agent.

The *dispatcher* agent may dispatch another agent to an external host. Thus a malicious agent may dispatch its partner malicious agent to an external host from the agent server.

The *migrator* agent simply migrates to another host from the server and takes

away the collected private information with it.

The *retracting agent* collects private information and waits for retraction. The originator retracts the agent from the server along with the private information. Retraction can be said as an indirect migration process.

In our experiments, we found that the privacy manager blocked every types of forbidden operations according to the security policies, which defines granted permissions, and proved its effectiveness in protecting malicious activities. An operation may require several permissions and the lack of any one of them can protect an agent from performing that operation. However, the privacy manager shows only the first permission which the agent lacks and which it encounters in the permission check (Table 5.6).

In web browsers, the hostile applets can send the acquired data to their origin hosts, because the browsers allow any downloaded applet to connect back to its origin host. iCOP does not allow the the mobile agents to connect to the outside world or to access to local resources through which they may transfer the acquired private information.

5.3.2 Privacy Loss

We measured the privacy loss through mathematical analysis and compared the privacy loss in our mechanism with those in traditional mechanisms for solving the vector dominance problem [51] and the meeting scheduling problem [66]. For the vector dominance problem, each participant solved the problem with a centralized algorithm separately by exchanging their private vectors with commitment protocol. For the meeting scheduling problem, the participants used a centralized algorithm such as standard (centralized) backtracking algorithm or a distributed backtracking algorithms such as ADOPT [46]. Privacy loss was measured using the Min metric [32] and the VPS metric [41]. The privacy loss of the system is taken as the average privacy loss of all the participants.

For the vector dominance problem, the agents exchange their vectors using a commitment protocol [49] from within the iCOP, resulting in both agents getting both vectors. Then, each agent simply compares the respective components of the two

vectors, creates the computational result, say "TRUE", (in the pre-defined format) and passes it to the service agent. Even though the participants get each other's private vectors from inside the iCOP, they cannot leak anything, i.e. no privacy loss.

With the meeting scheduling problem, a number of participants schedule a meeting with their private valuations of time at each slot (i.e., preferences) whose components can take values from the set $V := \{1, ..., K\}$. The value of K was varied from the set $\{3, 4, 5, 6, 7\}$. The number of time slots d was assumed to be 10 and the number of participants n was varied from 3 to 20.

The privacy loss in optAPO occurs in the initialization phase at which the participants exchange their private information with their neighbors. The minimum privacy loss (i.e., one neighbor) in optAPO is $d/((n-1)*d) = 1/(n-1)$ in the VPS metric and $n * 1/n = 1$ in the Min metric.

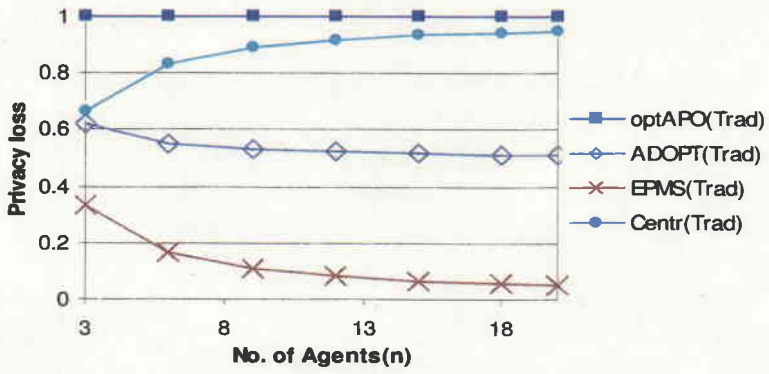
The privacy loss in the ADOPT algorithm varies from scenario to scenario and is calculated from the amount of information shared with the others. Thus, we measure the privacy loss in the ADOPT algorithm by taking the average privacy loss from different scenarios in the simulation.

In the EPMS algorithm, the privacy loss in the VPS metric is $1/(n*(n-1))$; $n > 2$ and that in the Min metric is d/n where, n is the number of participants [32].

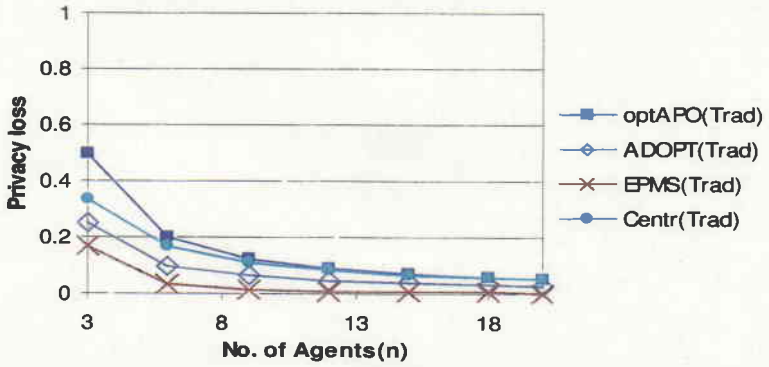
In the centralized algorithm, the privacy loss in the VPS metric is $(n-1)*(1/(n-1))/n = 1/n$ and that in the Min metric is $((n-1)*1)/n = (n-1)/n$ where, n is the number of participants. Note that we have taken into consideration that one of the participants solve the problem by accumulating all the inputs and using a centralized algorithm.

Figure 5.17(a) shows the privacy loss for varying numbers of participants in the four algorithms in the traditional architectures in the Min metric and Fig. 5.17(b) shows the same in the VPS metric. From the figure, we can see that in the traditional architecture, the privacy loss in the optAPO is the highest, followed by the centralized algorithm (in which one of the participants solves the problem), the ADOPT, and the EPMS.

The maximum privacy loss in the iCOP depends upon the inherent privacy loss in the algorithm and the maximum amount of information that can be leaked out from



(a) Min metric



(b) VPS metric

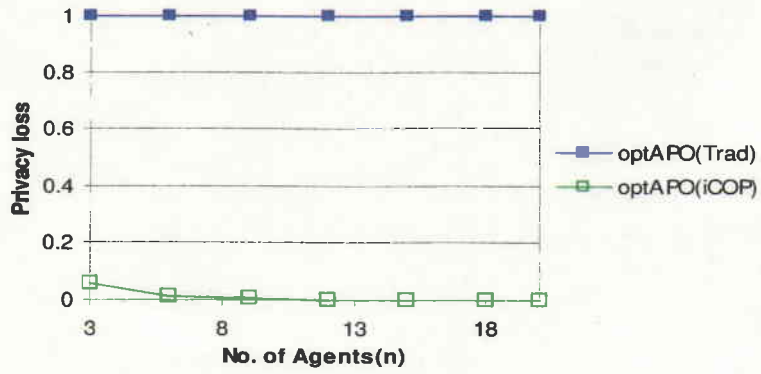
Figure 5.17: Relative privacy loss for four algorithms in traditional architecture.

the iCOP. For the purpose of analysis, the maximum privacy loss in the iCOP has been shown.

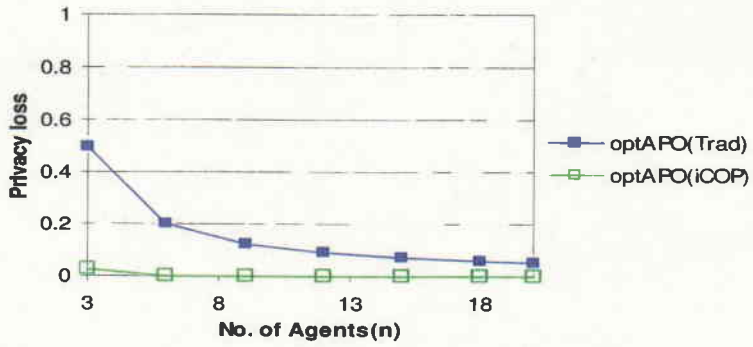
Figure 5.18(a) shows the average privacy loss for optAPO in the traditional architecture and in the iCOP for the Min metric and Fig. 5.18(b) shows the same in the VPS metric. We see that the privacy protection by the iCOP is significant.

Figures 5.19 and 5.20 show the average privacy loss in ADOPT and EPMS, respectively, and Fig. 5.21 shows the average privacy loss in the traditional centralized approach and multiple centralized algorithm by each participant.

The privacy losses in the iCOP security architecture using a distributed algorithm are much lower than those the algorithm causes in the traditional distributed architecture. We take into consideration that a number of possible solutions exist and a participating agent can bias all other participating agents towards a specific solution



(a) Min metric

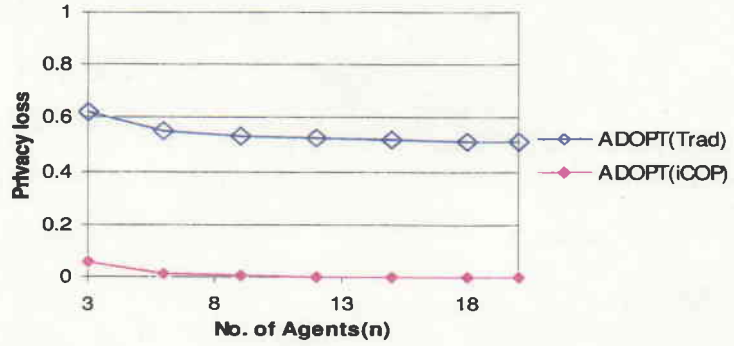


(b) VPS metric

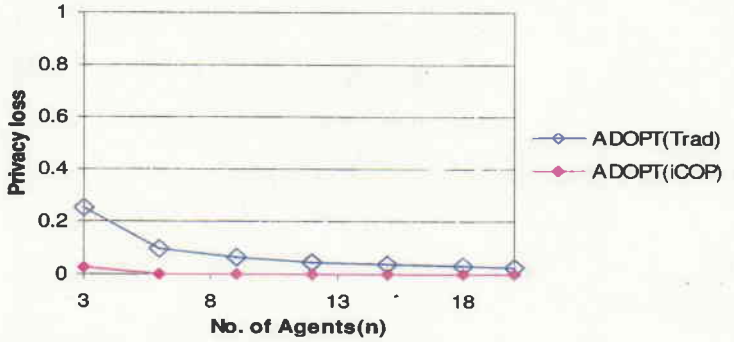
Figure 5.18: Reducing privacy loss in optAPO algorithm using iCOP.

by manipulating its own inputs. However, if there are fewer possible solutions, then the privacy loss in the iCOP architecture using a distributed algorithm will be less than that shown in the graph. Finally, the privacy loss in the iCOP security architecture using a centralized algorithm executed by each participant separately (i.e., multiple centralized algorithm) with the committed inputs resulted in no privacy loss (Fig. 5.21).

The privacy loss in the VPS metric is much lower than that in the Min metric. This is due to the differences in the definitions of the two metrics [32] [41]. The Min metric accounts for the number of participants that lose their information to any of the other participants, regardless of how many participants the information is revealed to. However, when the privacy is measured considering the fraction of the participants to whom the information is lost, as in the VPS metric, the privacy



(a) Min metric



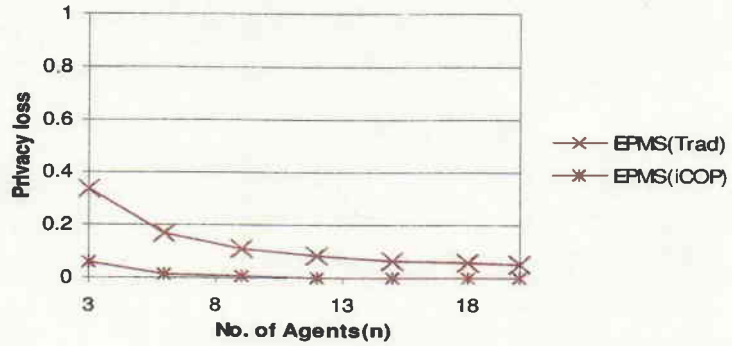
(b) VPS metric

Figure 5.19: Reducing privacy loss in ADOPT algorithm using iCOP.

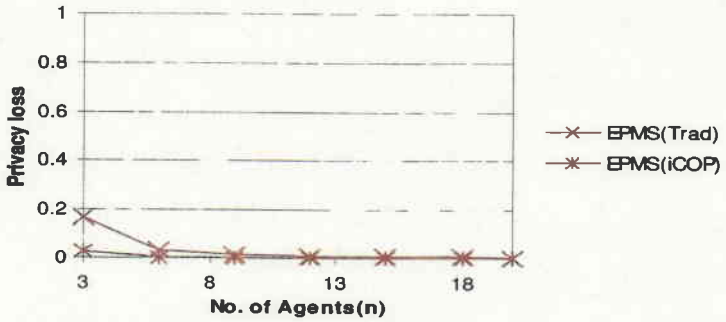
loss becomes much less, because the loss does not occur to all of the $(n-1)$ other participants.

For the distributed algorithm in the iCOP security architecture, the same information is revealed to other agents as for it in the distributed architecture, but within the iCOP, and only a very small fraction of the revealed information can be disclosed outside the iCOP, resulting in very low privacy loss. On the other hand, with each of the participants using centralized algorithms, no information can be disclosed outside the iCOP, resulting in no privacy loss.

We took into consideration that a number of possible solutions exist and a participating agent can bias all other participating agents towards a specific solution by manipulating its own inputs. However, if there are fewer possible solutions, then the privacy loss in the iCOP architecture using a distributed algorithm will be less than



(a) Min metric



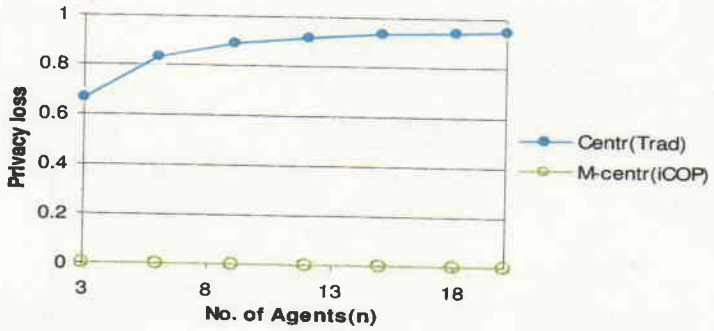
(b) VPS metric

Figure 5.20: Reducing privacy loss in EPMS algorithm using iCOP.

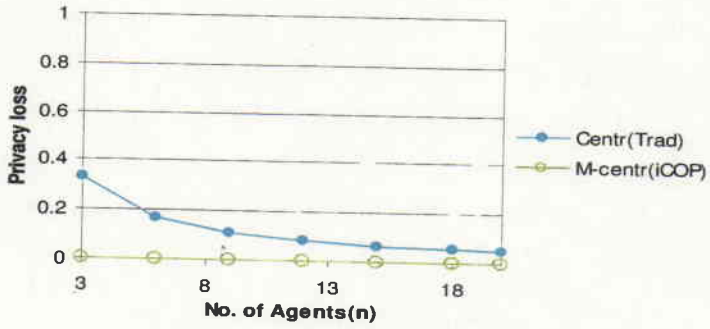
that shown in the graph.

5.3.3 Computational Time

A distributed algorithm in the iCOP results in very low privacy loss compared to that algorithm executed in the distributed architecture. On the other hand, a centralized algorithm executed individually in the iCOP by all of the participants results in no privacy loss. Executing a distributed algorithm in a server may require higher computational time than executing it in a distributed architecture. Also, the execution of a centralized algorithm by multiple agents in a server (as in our scheme) will require a much higher computational time than executing it once (as in a traditional centralized scheme). For the purpose of analysis, we consider the worst case, i.e. only one



(a) Min metric



(b) VPS metric

Figure 5.21: Reducing privacy loss by using multiple centralized algorithm in iCOP.

agent server exists in the iCOP domain. Then, we measure the computational time in the iCOP and compare it with the computational time in traditional architecture for the same algorithm.

For computational time measurements, the Cycle-based Runtime (CBR) metric [8] was used, which takes into consideration the concurrency in distributed algorithms and the latency of the underlying communication environment. To measure inter-agent remote messaging delays (in distributed architecture) and inter-platform agent migration delays (required in our mechanism), we used two hosts (Host A and Host B) with the specifications shown in Table 5.7 and the Aglet environment [33].

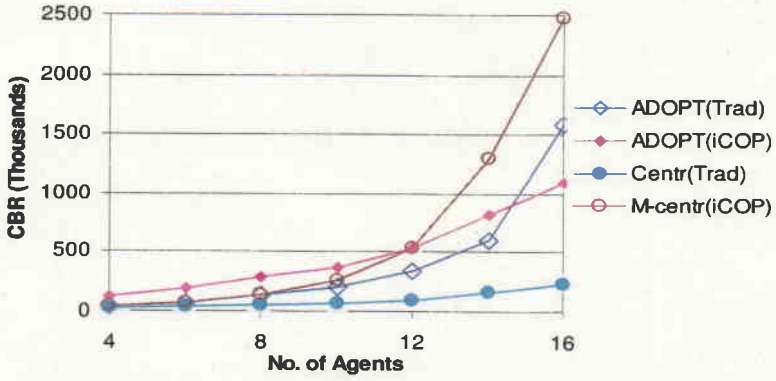
For a 23KB agent, it took about 21 ms for a single migration (including the serialization, de-serialization, class loading etc.) between two platforms, and the single messaging latency between the two agents at two platforms was 4 ms. The

Table 5.7: Specification of the hosts' used for measuring remote messaging time and migration time.

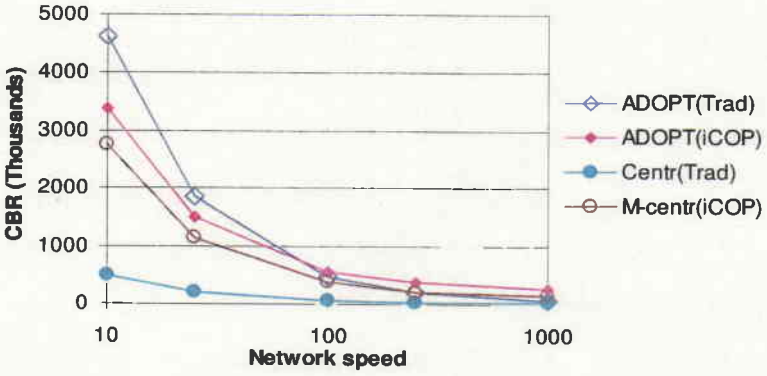
	Host A	Host B
Processor	1.4 GHz	1.8 GHz
RAM	768 MB	256 MB
NIC	100 Mbps	100 Mbps
OS	Windows XP	Windows XP

value of the messaging/migration time was represented in terms of the number of (concurrent) constraint checks. We also considered that one constraint check requires no more than 100 language level key instructions. From the measurement in our environment, single messaging time = 5000 constraint checks and agent migration time = 26250 constraint checks.

Figure 5.22(a) shows the relative computational times (taken averages of 15 runs) of the ADOPT algorithm in the distributed architecture, the ADOPT algorithm in the iCOP architecture, the centralized algorithm executed by a single agent, and the centralized algorithm executed by each of the participating agents in the iCOP architecture for varying numbers of agents/participants. The input values to the algorithm were taken randomly from a specific range. The complexities of the DCOP algorithms (e.g. ADOPT) become exponential with the number of agents because of their backtracking mechanisms. Computational time in the iCOP includes the agent migration time and the algorithm execution time at the server, while that in the distributed architecture includes remote messaging time and the algorithm execution time. From the Fig. 5.22(a), we see that for a small number of participants, the traditional centralized mechanism requires the least computational time, followed by the distributed algorithm executed in the traditional distributed architecture, the centralized algorithm executed by each of the participants in the iCOP, and the distributed algorithm in the iCOP. With the increase in the number of agents, the growth of computational time is minimized in the centralized mechanism, followed by the distributed algorithm in the iCOP, the distributed algorithm in the distributed architecture, and the centralized algorithm executed by each participant in the iCOP. Even though the single agent migration time in our mechanism is relatively larger



(a) For different number of agents.



(b) For different network speed.

Figure 5.22: Comparison of computational times in traditional architecture and in iCOP using CBR metric.

than a single messaging time in the distributed mechanism, the total agent migration (performed once) time in our mechanism is small as compared with the time for a larger number of remote messages in the distributed mechanism for large number of participants. However, in problems which involve only few a messages per agent, the distributed mechanism would require fewer CBR. Finally, a centralized algorithm executed by all of the participants separately at a single host requires the highest computational time, when the number of participants is very large.

Figure 5.22(b) shows the computational time (10 agents) with different relative network speeds L. The value of L for the network used in our experiment has been labelled 100. If the communication latency is decreased i.e., if we use a faster network, say $L = 1000$, then the computational time in the distributed mechanism becomes

lower than that in the iCOP. So, in a very high speed network, even for a large number of agents, the larger inter-agent messaging time in the distributed mechanism becomes very low and the serial execution time for all agents in our mechanism becomes high and as a result, the computational time in the distributed mechanism becomes lower than that in our mechanism. However, if we use a slower network, say $L = 10$, then the computational time in the iCOP becomes lower than that in the distributed mechanism.

The iCOP provides very good privacy in using distributed algorithm and complete privacy in using multiple centralized algorithms. For small number of participants, the computational times are almost the same in the two cases. However, for large number of participants the computational time in using a distributed algorithm in the iCOP is lower (Fig. 5.22(a)). Thus, for very sensitive data, multiple centralized algorithm should be used and for less sensitive data, distributed algorithm can be used, especially for large number of participants.

5.4 Discussion

The iCOP security architecture is similar to the Java Sandbox architecture. However, the default policies of the Sandbox architecture allow its agents to connect back to the originator [23], i.e. open channels between the Sandbox and the agent originator exist. This problem cannot be solved by simply closing all the Sandbox channels, because if this is done, the computational result cannot be sent to the users. In addition, the necessary message permission for the participating agents also creates additional permissions (e.g. socket permission). Due to the additive nature of Java permissions [23], when message permission is granted to the participating agents, access to the network sockets cannot be prevented in the default Sandbox. But by customizing the Java security manager, we can precisely control (restrict) the access to every resources. In addition, the service agent and the security policies of our protection mechanism help in sending only the computational result to the users protecting the shared private information.

The traditional centralized scheme for solving the multi-party computation problem requires disclosing all the private inputs of the participants and the computational

result to the central agent, who solves the problem on behalf of the participants. On the other hand, in our scheme, the participants solve the problem by themselves without disclosing their private information to a central agent and any other unauthorized party. One key advantage of using the proposed protection mechanism is that the maximum privacy loss is very low for any algorithm. Thus, existing simplest algorithm can be used.

5.5 Conclusion

We have presented a new privacy protection mechanism in solving multiparty computation problems without using complex cryptographic algorithms. In the proposed mechanism, the participating agents can use existing distributed algorithms or centralized algorithms to solve the problem, but in a confined agent platform named iCOP. In using a distributed algorithm, only a few bits may be leaked through the result sending channel depending upon the problem characteristics. The maximum privacy loss in using a distributed algorithm in our mechanism is much lower than the privacy loss caused by the algorithms in traditional distributed architecture. Also, our mechanism can provide complete privacy if the participants exchange their inputs with a commitment protocol and each of them solve the problem independently using centralized algorithms with the committed inputs within the iCOP. However, this requires higher computational time than for a distributed algorithm when the number of participants is large and a single server is used. The computational time can be reduced by load balancing i.e., by distributing the participating agents into different servers of the iCOP domain.

Chapter 6

Applications

6.1 Applications of the EPMS Algorithm

The EPMS algorithm has been evaluated for the MS problem. However, it can be used for other distributed scheduling problems, such as distributed resource allocation problems [18], distributed scheduling problems [59], and nurse time-tabling tasks [57]. For example, in the distributed resource allocation problem, each agent can be treated as a resource. Each task requires a number of resources, i.e. a number of participating agents. The resources have their constraints (e.g., they have previously fixed schedules). Allocating all of the required resources for a task is equivalent to scheduling a meeting. In our future work, we plan to evaluate the EPMS algorithm for the above-mentioned scheduling applications. We also plan to reduce privacy loss by reducing the value of probability of disclosure $\mathbb{P}_d$ in the agent server.

6.2 Applications of the Server-Assisted Privacy Protection Mechanism

Besides the application scenarios described in previous chapters, there are many other applications for which our proposed server-assisted privacy protection mechanism can be used. Following we briefly describe few applications.

Privacy-preserving Match-making The match-making problem [24] is an interesting problem in which privacy is an important issue. Suppose Alice and Bob want to find out whether they are interested in each other by matching their interests. The protocol has to be such that if both parties are found to be interested in each other, they will know the result, otherwise none of them should learn anything about the other party.

Privacy-preserving Cooperative Scientific Computations One such problem can be described as: Alice has m private linear equations represented by $M_1(x) =$

b_1, and Bob has $n - m$ private linear equations represented by $M_2(x) = b_2$, where x is an n-dimensional vector. Alice and Bob want to find a vector x that satisfies both of Alice's and Bob's equations.

The linear systems of equations problem, the linear least squares problem, and the linear programming problem have proved valuable for modeling many and diverse types of problems in planning, routing, scheduling, assignment, and design. In many cases, those linear equations or linear requirements are proprietary data and are too valuable to disclose to anybody else, especially to a potential competitor.

For instance, two financial organizations plan to cooperatively work on a project for mutual benefit. Each of the organizations would like its own requirements being satisfied (usually, these requirements are modeled as linear equations or linear inequalities). However, most of their requirements are proprietary data which includes interest and inflation rates, economic statistics, portfolio holding, etc. Therefore, nobody likes to disclose its requirements to the other party, or even to a third party. How could they cooperate on this project that has to satisfy everybody's private requirements without compromising the privacy requirements? The current practice is to operate "in the clear", that is, by revealing requirements to the other party or to the agent performing the computation. The consequence is obvious if the other party or the agent is not trusted.

Privacy-Preserving Database Query Suppose, Alice has a string q, and Bob has a database of strings $T = t_1....t_N$; Alice wants to know whether there exists a string t_i in Bob's database that "matches" q. The "match" could be an exact match or an approximate (closest) match. The privacy requirement is that Bob cannot know Alice's secret query q or the response to that query, and Alice cannot know Bob's database contents except for what could be derived from the query result.

Privacy-Preserving Data Mining Alice has a private database D_1, and Bob has a private database D_2. They want to jointly identify association rules in the union of D_1 and D_2.

For example, country A's intelligence agents have observed the activities $X = (x_1...x_n)$ for a period of time, and Country B's intelligence agents have observed the activities $Y = (y_1....y_m)$ for the same period of time. They want to collaboratively

find out whether the activities in Y has any correlation with the activities in X. The results of collaboration could help both countries to understand the trend of the behaviors of the target, such as the behaviors of some suspected terrorism organization, the military movement of a dangerous country, etc. However, neither A or B is willing to disclose its observation to the other countries because they don't fully trust each other. It is possible that B might use A's intelligence information (or sell it to the target) to uncover A's agents, and thus causing damage to A's intelligence agents.

Privacy-Preserving Geometric Computation Alice has a private shape a, and Bob has another private shape b; they both want to know whether a and b intersect. Alice does not want Bob or anybody else know any information about the shape a, nor does Bob want to disclose information about his shape b. Moreover, in no case should anybody learn the relative position between a and b, and, if these two regions intersect, nobody should learn where they intersect with each other.

One application of the above problem could be: Two companies plan to expand their market shares in certain regions, but, they do not want to compete in the same region. What they really want to know is whether their selected regions overlap with each other. Because the information about its own selected region is so valuable to each company, neither of them wants to disclose it to the other party.

Privacy-Preserving Statistical Analysis This problem has a lot of applications. For example, a bank wants to investigate if ages can affect people's financial activities. However, the bank only has customers' financial activities, it does not know the ages of its customers. Therefore, the bank turns to some government bureau who has the knowledge of every person's dates of birth, but the government bureau is required by laws not to disclose it. On the other hand, the customers' financial activities are the bank's proprietary data that the bank does not want to disclose to anybody.

In another example, a school wants to investigate the relationship between people's intelligence quotient (IQ) score and their annual salary. The school has its students' IQ score, but does not have students' salary information; therefore the school needs to cooperate with companies that hire the students, but those companies are not willing to disclose the salary information. On the other hand, the school cannot give

students' IQ score to their employers either. A privacy-preserving statistical analysis method [12] is needed to solve this problem.

Selection problem (select median, select the k^{th} smallest element): Alice has a private data set d_1, and Bob has a private data set d_2; they want to find the median (or the k^{th} smallest element) among the data in $d_1 \cup d_2$.

Sorting problem: Alice has a private data set d_1, and Bob has a private data set d_2; they want to sort the elements in the union of these two data sets $d_1 \cup d_2$, such that each element in these two data sets is marked by a number representing the order of this element.

Chapter 7

Conclusions and Future Works

7.1 Summary of Main Results

Firstly, we have devised a privacy loss model for multi-party computation problems and shown the factors which affect the amount of privacy loss. Then, we have proposed a metric for measuring privacy loss qualitatively. We have also presented a mobile agent-based scheduling scheme, called EPMS, and evaluated it with respect to computational complexity, privacy loss, and global utility for the meeting scheduling (MS) problem. The algorithm has a polynomial time complexity for scheduling multiple meetings concurrently. We measured its schedule privacy loss and preference privacy loss for the MS problem according to the VPS privacy metric and Min privacy metric. We also compared the privacy loss in EPMS with that in a well-known optimization algorithm, optAPO, and that in the centralized algorithm. The schedule privacy loss in the VPS privacy metric is the minimal in the centralized algorithm, while in the Min privacy metric, it is the minimal in EPMS. The preference privacy loss in EPMS is the minimal in both the VPS privacy metric and the Min privacy metric. The global utility in EPMS was found to be above 98% of that in the optimal solution, for scheduling smaller numbers of meetings concurrently, but decreases slowly for larger numbers of meetings. Thus, we can say that EPMS results in a good tradeoff among complexity, privacy, and global utility for solving the scheduling problem.

We have also presented a new privacy protection mechanism in solving multi-party computation problems without using complex cryptographic algorithms. In the proposed mechanism, the participating agents are trapped into a confined agent platform, named iCOP, where they can use existing (distributed or centralized) algorithms for solving the problem. In using a distributed algorithm, only a few bits may

be leaked out through the result sending channel depending upon the problem characteristics. The maximum privacy loss in using a distributed algorithm in iCOP is much lower than the privacy loss caused by the algorithms in traditional distributed architecture. Also, our mechanism can provide complete privacy protection if the participants exchange their inputs with a commitment protocol and each of them solve the problem independently with centralized algorithms within the iCOP using the committed inputs. However, this requires higher computational time than using a distributed algorithm when the number of participants is large and a single server is used. The computational time can be reduced by load balancing i.e., by distributing the participating agents into multiple servers of the iCOP domain.

7.2 Future Works

In this thesis, we have evaluated the EPMS algorithm for the MS problem. In our future work, we plan to evaluate the EPMS algorithm for other scheduling problems, such as distributed resource allocation problems [18], distributed scheduling problems [59], and nurse time-tabling tasks [57].

Currently, iCOP deals with the type of multi-party problems in which the same computational result can be disclosed to all of the users. In other words, we considered that the disclosure of the same computational result to all of the participants does not causes privacy loss. However, there are some applications in which different participants should have different computational result and the disclosure of the same computational result to all of them results in privacy loss. For example, in e-commerce applications such as virtual marketplace, it is needed that different private information should be kept secret from different parties. For example a buyer may want to keep her postal address secret from the vendor and the item information secret from the delivery company. In our future work, we plan extend iCOP for protecting privacy in solving such applications.

Data integration, sharing, and mining the integrated data from distributed, heterogeneous, and autonomous sources in order to discover important knowledge have been a long standing challenge for the database and data mining communities. The

increasingly exponential growth of distributed personal data could fuel data integration applications to address real life and more importantly life threatening problems such as efficient disease control. However, these important activities have also raised legitimate and widespread concerns of data privacy. We want to take into account such more practical privacy protection problems.

Privacy risk is the business risk resulting from the collection, use, retention, and disclosure of personal information. Like all business risk privacy risk could result in a loss of revenue or increased cost. Privacy risk management is organizational actions that control privacy risk. Privacy risk management is most often a pragmatic approach to reducing privacy threats. Three most common privacy risk management are: (a) advice provided by a privacy officer to business units upon requests, typically in support of a project or a corporate initiative, (b) Proactive initiatives (such as personal information handling policies, employee training and audits) implemented corporate-wide to reduce privacy risk, initiated by the privacy office, (c) a privacy officer and a business unit representative go through processes, often during an audit process or in support of a project, such as identifying privacy risk, assessing current risk management measures and developing new strategies to mitigate the identified risks. We want to address the risk management issue in our future works.

Finally, server security and trust management among the servers in the iCOP domain is an important security issue. What type of relationship among the servers should be maintained, how they should be configured to maintain robust security etc. are critical concerns. Also, detection of unusual suspicious events and their preventive measures are important. In our future works, we want to address these issues.

Bibliography

[1] M.J. Atallah and W. Du, *Secure multi-party computational geometry*, Proc. of the Seventh Int. Workshop on Algorithms and Data Structures, 2001, pp. 165–179.

[2] D.E. Bakken, R. Parameswaran, D.M. Blough, A.A. Franz, and T.J. Palmer, *Data obfuscation: anonymity and desensitization of usable data sets*, IEEE Security & Privacy **2** (2004), no. 6, 34–41.

[3] K. Bennett, *Linguistic steganography: survey, analysis, and robustness concerns for hiding information in text*, Tech. Report 2004-13, CERIAS.

[4] J. Biskup and U. Flegel, *On pseudonymization of audit data for intrusion detection*, Workshop on Design Issues in Anonymity and Unobservability, 2000, pp. 161–180.

[5] ———, *Transaction-based pseudonyms in audit data for privacy respecting intrusion detection*, Proc. of the Third Int. Workshop on Recent Advances in Intrusion Detection, 2000, pp. 28–48.

[6] G. Brassard, C. Crepeau, and J. Robert, *All-or-nothing disclosure of secrets*, Lecture Notes in Computer Science, no. 263, 1987, pp. 612–613.

[7] C. Cachin, *Efficient private bidding and auctions with an oblivious third party*, Proc. of the 6th ACM Conf. on Computer and communications security, 1999, pp. 120–127.

[8] J. Davin and P.J. Modi, *Impact of problem centralization in distributed constraint optimization algorithms*, Fourth Int. Joint Conf. on Autonomous Agents and Multiagent Systems, 2005, pp. 1057–1063.

[9] Y. Desmedt, *Some recent research aspects of threshold cryptography*, Lecture Notes in Computer Science, no. 1196, 1997, pp. 158–173.

[10] W. Du, *A study of several specific secure two-party computation problem*, Ph.D. thesis, Purdue University, USA, 2001.

[11] W. Du and M.J. Atallah, *Privacy-preserving cooperative scientific computations*, Proc. of the 14th IEEE Workshop on Computer Security Foundations, 2001, pp. 273–282.

[12] ———, *Privacy-preserving statistical analysis*, Proc. of the Seventeenth Annual Computer Security Applications Conf., 2001, pp. 102–110.

[13] L.D. Erman, F. Hayes-Roth, V.R. Lesser, and D.R. Reddy, *The hearsay-ii speech understanding system: integrating knowledge to resolve uncertainty*, Computing Surveys **12** (1980), no. 2, 213–253.

[14] S. Even, O. Goldreich, and A. Lempel, *A randomized protocol for signing contracts*, Communications of the ACM **28** (1985), 637–647.

[15] M.S. Franzin, E.C. Freuder, F. Rossi, and R. Wallace, *Multi-agent meeting scheduling with preferences: Efficiency, privacy, loss, and solution quality*, Proc. of the Eighteenth American Association for Artificial Intelligence Conf., 2002, pp. 25–32.

[16] E.C. Freuder, M. Minca, and R.J. Wallace, *Privacy/efficiency tradeoffs in distributed meeting scheduling by constraint-based agents*, Proc. of Int. Joint Conf. on Artificial Intelligence, 2001, pp. 63–72.

[17] P. Gemmell, *An introduction to threshold cryptography*, CryptoBytes **2** (1997).

[18] K. Ghedira and G. Verfaillie, *A multi-agent model for the resource allocation problem: a reactive approach*, Proc. of the Tenth European Conf. on Artificial Intelligence, 1992, pp. 252–254.

[19] V. D. Gligor, *A guide to understanding covert channel analysis of trusted systems*, Tech. Report NCSC-TG-030, American National Computer Security Center, 1993.

[20] V.D. Gligor, *A guide to understanding trusted facility management*, Tech. Report NCSC-TG-015, American National Computer Security Center, 1989.

[21] O. Goldreich, *Secure multi-party computation*, Available from http://www.wisdom.weizmann.ac.il/oded/PS/prot.ps, 2002.

[22] S. Goldwasser, *Multi-party computations- past and present*, Proc. of Sixteenth Annual ACM Symposium on Principles of Distributed Computing, 1997, pp. 1–6.

[23] L. Gong, G. Ellison, and M. Dageforde, *Inside java 2 platform security: Architecture, api design, and implementation*, second ed., Addison-Wesley, 2003.

[24] V. Graaf, *The matchmaking problem*, CWI Quarterly **8** (1995), no. 2, 129–146.

[25] R. Greenstadt, J.P. Pearce, E. Bowring, and M. Tambe, *Experimental analysis of privacy loss in dcop algorithms*, Proc. of Fifth Int. Joint Conf. on Autonomous Agents and Multiagent Systems, 2006.

[26] A.B. Hassine, X. Defago, and T.B. Ho, *Agent-based approach to dynamic meeting scheduling problems*, Proc. of the Int. Conf. on Autonomous Agents & Multiagent Systems, 2004, pp. 1132–1139.

[27] T. Haynes, S. Sen, N. Arora, and R. Nadella, *An automated meeting scheduling system that utilizes user preferences*, Proc. of the First Int. Conf. on Autonomous Agents, 1997, pp. 308–315.

[28] T. Herlea, J. Claessens, G. Neven, F. Piessens, B. Preneel, , and B.Decker, *On securely scheduling a meeting*, Proc. of IFIP Int. Information Security Conf., 2001, pp. 183–198.

[29] M. Hirt and U. Maurer, *Robustness for free in unconditional multi-party computation*, Lecture Notes in Computer Science, vol. 2139, 2001, pp. 101–118.

[30] M. Hirt and J.B. Nielsen, *Upper bounds on the communication complexity of optimally resilient cryptographic multiparty computation*, Lecture Notes in Computer Science, vol. 3788, 2005, pp. 79–99.

[31] M.N. Huda, E. Kamioka, and S. Yamada, *A mobile agent based computing model for enhancing privacy in multi-party collaborative problem solving*, Proc. of the Third Int. Conf. on Mobile Computing and Ubiquitous Networking, 2006, pp. 107–115.

[32] ———, *An efficient and privacy-aware meeting scheduling scheme using common computational space*, IEICE Trans. on Information & Systems **E90-D** (2007), no. 3.

[33] IBM, *Aglet software development kit*, http://sourceforge.net/projects/aglets/.

[34] Javvin Technologies Inc, *Network packet analyzer*, http://www.javvin.com/packet.html.

[35] L. Ismail and D. Hagimont, *A performance evaluation of the mobile agent paradigm*, Proc. of the Int. Conf. on Object Oriented Programming, Systems, Languages, and Applications, 1999, pp. 306–313.

[36] R. L. Kay, *How to implement trusted computing: a guide to tighter enterprise security*, Tech. report, Trusted computing group.

[37] R.A. Kemmerer, *A practical approach to identifying storage and timing channels: twenty years later*, Proc. of Eighteenth Annual Computer Security Applications Conf., 2002, pp. 109–118.

[38] M. LaDue's, *Hostile applets source code*, http://www.cigital.com/hostile-applets/SourceCode.html.

[39] D.B. Lange and M. Oshima, *Programming and deploying java mobile agents with aglets*, second ed., Addison-Wesley, 1998.

[40] Y. Lindell and B. Pinkas, *Privacy preserving data mining*, Lecture Notes in Computer Science, vol. 1800, 2000, pp. 20–24.

[41] R.T. Maheswaran, J.P. Pearce, E. Bowring, P. Varakantham, and M. Tambe, *Privacy loss in distributed constraint reasoning: a quantitative framework for analysis and its applications*, Journal of Autonomous Agents and Multi-Agent Systems **13** (2006), no. 1, 27–60.

[42] R.T. Maheswaran, J.P. Pearce, P. Varakantham, E. Bowring, and M. Tambe, *Valuations of possible states (vps)- a quantitative framework for analysis of privacy loss among collaborative personal assistant agents*, Proc. of Fourth Int. Joint Conf. on Autonomous Agents and Multiagent Systems, 2005, pp. 1030–1037.

[43] R.T. Maheswaran, M. Tambe, E. Bowring, J.P. Pearce, and P. Varakantham, *Taking dcop to the real world: efficient complete solutions for distributed multi-event scheduling*, Proc. of the Third Int. Joint Conf. on Autonomous Agents and Multiagent Systems, 2004, pp. 310–317.

[44] R. Mailler and V. Lesser, *Solving distributed constraint optimization problems using cooperative mediation*, Proc. of the Third Int. Joint Conf. on Autonomous Agents and Multiagent Systems, 2004, pp. 438–445.

[45] S. T. Margulis, *Conceptions of privacy: Current status and next steps*, Journal of Social Issues **33** (1977), no. 3, 5–21.

[46] P.J. Modi, W. Shen, M. Tambe, and M. Yokoo, *Adopt: asynchronous distributed constraint optimization with quality guarantees*, Artificial Intelligence Journal **161** (2005), 149–180.

[47] M. Naor and B. Pinkas, *Oblivious transfer and polynomial evaluation (extended abstract)*, Proc. of the Thirty-first ACM Symposium on Theory of Computing, 1999, pp. 245–254.

[48] M. Rabin, *How to exchange secrets by oblivious transfer*, Tech. Report Memo TR-81, Aiken Computation Laboratory, 1981.

[49] J.C. Roca and J.D. Ferrer, *A non-repudiable bitstring commitment scheme based on a public-key cryptosystem*, Proc. of Int. Conf. on Information Technology: Coding and Computing, 2004.

[50] R. Rosenberg, *The social impact of computers*, Academic Press, 1992.

[51] Y. Sang, H. Shen, and Z. Zhang, *An efficient protocol for the problem of secure two-party vector dominance*, Proc. of Int. Conf. on Parallel and Distributed Computing, Applications and Technologies, 2005.

[52] B. Schneier, *Applied cryptography: Protocols, algorithms, and source code in c*, John Wiley & Sons, Inc., 1996.

[53] S. Sen, T. Haynes, and N. Arora, *Satisfying user preferences while negotiating meetings*, Int. Journal of Human-Computer Studies **47** (1997), no. 3, 407–427.

[54] A. Shamir, *How to share a secret*, Communication of the ACM **22** (1979), no. 11, 612–613.

[55] M.C. Silaghi and D. Mitra, *Distributed constraint satisfaction and optimization with privacy enforcement*, Proc. of the IEEE/WIC/ACM Int. Conf. on Intelligent Agent Technology, 2004, pp. 531–535.

[56] B. Snow, *Four ways to improve security*, IEEE Security & Privacy Magazine **3** (2005), no. 3, 65–67.

[57] G. Solotorevsky and E. Gudes, *Solving a real-life time tabling and transportation problem using distributed csp techniques*, Proc. of Workshop on Constraint Programming Applications, 1996, pp. 123–131.

[58] L. Sweeney, *K-anonymity: A model for protecting privacy*, Int. Journal of Uncertainty, Fuzziness and Knowledge-Based Systems **10** (2002), no. 5, 557–570.

[59] K. Sycara, S. Roth, N. Sadeh, and M. Fox, *Distributed constrained heuristic search*, IEEE Tran. on Systems, Man and Cybernetics **21** (1991), no. 6, 1446–1461.

[60] Z. B. Trubiniova and M. Hirt, *Efficient multi-party computation with dispute control*, Lecture Notes in Computer Science, vol. 3876, 2006, pp. 305–328.

[61] T. Tsuruta and T. Shintani, *Scheduling meetings using distributed valued constraint satisfaction algorithm*, Proc. of European Conf. on Artificial Intelligence, 2000, pp. 383–387.

[62] S. D. Warren and L. D. Brandeis, *The right to privacy*, Harvard Law Review **4** (1890), no. 5, 193–220.

[63] A. F. Westin, *Privacy and freedom*, Atheneum, 1967.

[64] A. Yao, *Protocols for secure computations*, Proc. of the 23rd Annual IEEE Symposium on Foundations of Computer Science, 1982.

[65] M. Yokoo, E.H. Durfee, T. Ishida, and K. Kuwabara, *The distributed constraint satisfaction problem: formalization and algorithms*, vol. 10, 1998, pp. 673–685.

[66] M. Yokoo, K. Suzuki, and K. Hirayama, *Secure distributed constraint satisfaction: reaching agreement without revealing private information*, Proc. of the Eighth Int. Conf. on Principles and Practice of Constraint Programming, 2002, pp. 387–401.

[67] S. Zhu, S. Setia, and S. Jajodia, *Performance optimizations for group key management schemes*, Proc. of the 23rd Int. Conf. on Distributed Computing Systems, 2003, pp. 163–171.