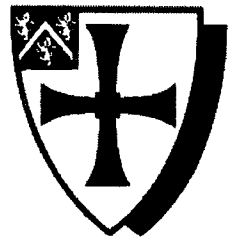
Software-implemented attack tolerance for critical information retrieval

Yunwen (Erica) Yang

Software-Implemented Attack Tolerance for Critical Information Retrieval

Yunwen (Erica) Yang

杨云雯



Submitted in partial fulfilment of the requirement for the degree of Doctor of Philosophy

Department of Computer Science
University of Durham
United Kingdom

December 2004

This research was supported by a Teaching Assistantship from the University of Durham and the e-Demand project from the U.K. Engineering and Physical Sciences Research Council (EPSRC) and the Department of Trade and Industry (DTI) e-Science Core Programme under grant no. THBB/C008/00176C.

Software-Implemented Attack Tolerance for Critical Information Retrieval

Yunwen (Erica) Yang

Department of Computer Science
University of Durham

Thesis Abstract

The fast-growing reliance of our daily life upon online information services often demands an appropriate level of privacy protection as well as highly available service provision. However, most existing solutions have attempted to address these problems separately. This thesis investigates and presents a solution that provides both privacy protection and fault tolerance for online information retrieval. A new approach to Attack-Tolerant Information Retrieval (ATIR) is developed based on an extension of existing theoretical results for Private Information Retrieval (PIR). ATIR uses replicated services to protect a user's privacy *and* to ensure service availability. In particular, ATIR can tolerate any collusion of up to t servers for privacy violation and up to f faulty (either crashed or malicious) servers in a system with k replicated servers, provided that $k \geq t + f + 1$ where $t \geq 1$ and $f \leq t$. In contrast to other related approaches, ATIR relies on neither enforced trust assumptions, such as the use of tamper-resistant hardware and trusted third parties, nor an increased number of replicated servers. While the best solution known so far requires k ($\geq 3t + 1$) replicated servers to cope with t malicious servers and any collusion of up to t servers with an $O(n^{\frac{1}{\lceil ((k-1)/3t \rceil}})$ communication complexity, ATIR uses fewer servers with a much improved communication cost, $O(n^{\frac{1}{2}})$ (where n is the size of a database managed by a server).

The majority of current PIR research resides on a theoretical level. This thesis provides both theoretical schemes and their practical implementations with good performance results. In a LAN environment, it takes well under half a second to use an ATIR service for calculations over data sets with a size of up to 1MB. The performance of the ATIR systems remains at the same level even in the presence of server crashes and malicious attacks. Both analytical results and experimental evaluation show that ATIR offers an attractive and practical solution for ever-increasing online information applications.

Keywords: active attacks, attack tolerance, fault tolerance, malicious attacks, performance, private information retrieval, privacy protection, query services, and security.

Acknowledgements

This thesis officially marks the end of my five-year long study in Durham, a wonderful place in north east England where I will always be happy to go back to walk around. The river, the woods, the castle, and the Cathedral, I miss you all. Going to Durham is probably one of the most important career decisions I have ever made. It was there that I met so many nice people who have transformed my life all the way along. Among many, I would like to first express my gratitude to my supervisor, Professor Jie Xu, who made a great effort to secure me the funding in the first place, trained me to be a professional researcher, believed in me, and always inspired me to be the best. I would like to express my deep appreciation for his supervision, guidance, patience, and inspiration. These are the valuable gifts I shall enjoy for the rest of my life.

My appreciation also goes to Professor Malcolm Munro, for kindly being my supervisor during the last stage of my study, and for his encouragement and support.

Thanks to all the members of the Distributed Systems and Services Group, the Department of Computer Science, and my friends in Durham for numerous research discussions, jokes, trips, meals, and support. It was you guys who have made my life colourful, rich and enjoyable for the past five years.

I would also like to express deep thanks to my family, who have always been there helping, encouraging, loving, and being proud of me. Special thanks to Andrew, for putting up with me, for always being there for me, and for your belief in me, which I appreciate so much.

Table of Content

List of Figures

List of Tables

Acronyms

ATIR	Attack Tolerant Information Retrieval
cPIR	computational Private Information Retrieval
dATIR	deterministic Attack Tolerant Information Retrieval
hPIR	hardware-based Private Information Retrieval
LAN	Local Area Network
pATIR	probabilistic Attack Tolerant Information Retrieval
PIR	Private Information Retrieval
rPIR	robust Private Information Retrieval

Declaration

The material contained within this thesis has not previously been submitted for a degree at the University of Durham or any other university. The research reported within this thesis has been conducted by the author unless indicated otherwise.

Some parts of the work presented in this thesis have previously appeared in the following papers:

[YXB01] E. Y. Yang, J. Xu and K. H. Bennett, "*The SeCode Approach: Towards Fault-tolerant and Secure Execution of Mobile Code*", in Proc. IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'01), Goteborg, July 2001, Fast Abstract, Supplement, Goteborg, Sweden, 1-4 July 2001, pp. B74-B75.

[YXB02a] E. Y. Yang, J. Xu and K. H. Bennett, "*Private Information Retrieval in the Presence of Malicious Faults*", in Proc. 26th IEEE International Symposium on Computer Software and Applications (COMPSAC 2002), Oxford, UK, Aug. 2002, pp. 805-810.

[YXB02b] E. Y. Yang, J. Xu and K. H. Bennett, "*A Fault-Tolerant Approach to Secure Information Retrieval*", in Proc. 21st IEEE International Symposium on Reliable Distributed Systems (SRDS2002), Suita, Osaka, Japan, Oct. 2002, pp. 12-21.

[YXB03] E. Y. Yang, J. Xu, and K. H. Bennett, "*Sharing with Limited Trust: An Attack-Tolerant Service in Durham e-Demand Project*", in Proc. UK eScience 2nd All-Hands Meeting, Simon J. Cox Eds., Sept. 2nd-4th, 2003, Nottingham, U.K., ISBN 1-904425-11-9.

[YXB04] E. Y. Yang, J. Xu, and K. H. Bennett, "*Privacy-Enhanced Transactions for Virtual Organisations*", in Proc. UK e-Science 3rd All-Hands Meeting, Simon J. Cox Eds., 31st August – 3rd September, 2004, Nottingham, UK, ISBN 1-904425-21-6.

Chapter 1 Introduction

"Even when it is possible to build them, highly trustworthy components are costly."

Trust in Cyberspace
Fred B. Schneider, Editor

Committee on Information Systems Trustworthiness
National Research Council (USA), 1999.

1.1 Problem Statement

Querying online information services has gradually become an integral part of our daily life and work. The fast-growing development along this direction gives rise to significant security and reliability issues. Individuals become much more concerned about online privacy issues than ever. Our society, as a whole, becomes increasingly vulnerable to service disruptions. At the same time, the ever-increasing number of malicious attacks makes it even more difficult to maintain privacy protection and service provision in online services. There is an urgent need for techniques which can protect users' privacy as well as simultaneously ensure highly available services even in the presence of malicious attacks.

There has been considerable research effort put into online privacy protection. One such effort is Private Information Retrieval (PIR), an approach aiming to protect a user's privacy by keeping the intention of queries secret from curious database servers [CGKS95]. The principle of PIR is simple: instead of asking for one specific data item, a user queries an entire database. As a result, each data item is equally likely to be the one of interest, rendering the server unable to figure out the user's actual intention. PIR is based on a passive attack model where the server is deemed to be controlled by an attacker. In this model, an attacker only passively observes queries without modifying the execution flow of PIR protocols. This simplified assumption about attackers can easily be invalid due to the ever-increasing presence of malicious attacks. Once occupying the server (i.e. inside the trust perimeter of a system) and having gathered private information, a benign attacker can easily turn

into a malicious one. For example, due to the ease of gathering inside information, the attacker can easily forge bogus requests and cause denial-of-service attacks. This type of attacks is very difficult to detect and eliminate because such requests are not distinguishable from legitimate ones. The attacker may even control the server to deliver purposefully manipulated answers. Again, such answers are also indistinguishable from what a server is supposed to return in normal circumstances.

The majority of progress made along the line of PIR is based on a passive attack model [CGKS95, Amb97, BIM00, BI01, BIKR02, SS01, IS03, Aso04, IS04]. Various communication efficient schemes have been proposed to solve the PIR problem. In contrast, little is known about extending PIR in an active attack model, a more realistic model making little assumptions on the behaviours of faulty components. There are basically two approaches of dealing with malicious attackers: attack detection/prevention and attack tolerance. Traditionally, attack detection/prevention is heavily used to keep attackers outside a system, such as using cryptographic algorithms and firewalls. Due to the increasing size and complexity of computer systems, preventing attackers from hacking into a system becomes more costly and very difficult. Another approach for dealing with attacks is through the tolerance paradigm. Instead of purely preventing every single attack, a system is designed to tolerate attacks by allowing some components to be compromised and trigger mechanisms to ensure the delivery of correct services despite attacks [VNC03]. For example, by replicating a set of servers and using fault tolerant algorithms to identify correct results, the system can still deliver a correct service as promised in spite of the corruption of some of the servers. BFT [Cas01], COCA [ZRS02], and SINTRA [CP02] are three representative systems. All three systems are based on the state machine replication approach [Sch90] along the tolerance paradigm.

The majority of PIR research aims to demonstrate theoretical feasibilities and many obstacles preclude them from being used in practice. This is mainly due to two reasons: i) the standard database model of PIR schemes is only of theoretical interest (e.g. [SS01, YXB02a, YXB02b, IS03, Aso04, IS04]); hence, it is difficult to implement PIR schemes; and ii) PIR schemes are perceived to be computationally expensive (e.g. [BIM00, Kus03]). Initiated by IBM research in 2001 [SS01], researchers have begun to investigate practical PIR schemes such as those presented

in [YXB02a, YXB02b, YXB03, IS03, Aso04, IS04]. However, some of these systems [IS03, Aso04, IS04] build on the assumption that the database server will use secure hardware to correctly perform PIR operations. This assumption significantly restricts the scalability of these solutions and aggravates the computation problem of PIR. Some early performance studies (e.g. [SS01, IS03, Aso04]) show that this approach is expensive which in turn limits its practicability.

The thesis aims to tackle the Attack Tolerant Information Retrieval (ATIR) problem which is an extension of the PIR problem by taking malicious attacks into account.

1.2 Research Challenges

This section outlines the general research challenges of achieving ATIR while remaining practical to be implemented and be reasonably deployable. This will help to draw the research boundary of this thesis. Based on our understanding of the existing research on PIR and research on attack tolerance, the following five ATIR research challenges are derived.

1) Communication Complexity

Considerable research has been conducted on deriving communication efficient PIR schemes, such as those presented at [CGKS95, Amb97, BI01, BIKR02, BIM00, BS02]. This is a well-explored area. The problem and the basic approaches for tackling the challenge are relatively well understood. Recent studies show that some existing PIR schemes already have close to optimal communication complexity [WW04, Cha04]. In contrast, many other aspects of PIR, for example, the practical aspects, remain much less understood. Our research is focussed on those aspects by build on existing PIR schemes which already have a reasonable level of communication complexity.

2) Trust Assumptions

Existing PIR schemes often rely on unrealistic trust assumptions before and during the processing of PIR queries. The use of unrealistic trust assumptions restricts the applicability of these schemes. Existing attack-tolerant systems also make various degrees of trust assumptions on external trusted parties: a trusted third party is often required to securely and manually set up these systems. Trust assumptions are also

placed during the processing stages of all hardware-based PIR systems. For example, computation with secure hardware [SS01, IS03, AF02] in the system and the reliance on an external auxiliary server [GGM98] are the typical trust assumptions in current PIR research. With the use of these trust assumptions at various stages of deploying a system, the system is restricted to a static group membership. Dynamic addition and removal of members is costly and requires the involvement of trusted third parties (e.g. human operators). Consequently, such a system cannot cope with the dynamicity membership requirement of its environment.

3) Use of Replication

In PIR, replication is introduced as a fundamental means to reduce the communication complexity of unconditionally secure PIR schemes [CGKS95]. PIR schemes themselves do not provide mechanisms to support replication but rely on existing well-established replications protocols to disseminate updates and manage consistency among replicated servers. An inherent problem with replication is scalability, because replication protocols heavily rely on message passing to coordinate and synchronise among replicas. Replication should be used with care. In principle, the more replicated servers a PIR scheme uses the better communication complexity it can achieve. It becomes a challenge to find a balanced degree of replication which can provide reduced communication complexity while not being too costly.

4) Implementation

The standard PIR database model is too restrictive to be integrated with commercial database technologies. Most existing PIR schemes only demonstrate the theoretical feasibility of constructing such schemes, and little practical implementations of these schemes have been attempted. Extending the model is mandatory if the technique is to be used to implement a real service. Some PIR schemes and corresponding implementations use secure hardware to perform PIR operations. Due to its processing limitation, the performance shown by the secure-hardware based implementations [SS01, IS03, Aso04] is not satisfactory.

5) Processing Costs

It is often envisaged that PIR processing is computationally expensive, which is the main motivation for the work presented in [Kus03, IS03, SS01, AF02, BIM00]). Pre-

processing was introduced into various PIR schemes as a means to reduce online processing costs [BIM00, IS03, AF02, IS04, Aso04]. However, pre-processing is still costly: some recent experimental results show that the pre-processing time for secure hardware-based PIR schemes is in the order of several hours whereas the online processing time for retrieving one single record is in the order of several minutes [IS03, AF02, Aso04]. Hence, from a practical point of view, processing cost is still a major decisive factor determining the real impact of ATIR technologies. Although these implementations demonstrate the practicability of such PIR schemes, the performance shown by the results is far worse than the performance of state of the art database technologies.

1.3 Contributions

This thesis contributes to the state-of-the-art security and fault tolerance research by presenting a new approach for achieving an adjustable level of privacy protection whilst maintaining a high level of service provision in the presence of malicious attacks. Specifically, the thesis tackles a number of less understood open research challenges (challenges 2-5 presented in the previous section) in PIR research and provides a satisfactory answer to these challenges. In particular, this thesis contributes to the state-of-the-art PIR research and the state-of-the-practice practical PIR implementation in the following aspects.

ATIR requires limited trust assumptions on remote execution environment. With neither relying on secure hardware nor trusted third parties on remote execution environment, replication-based ATIR schemes are developed to protect the privacy of a user and ensure the provision of an information retrieval service. Unlike existing practical PIR approaches, the ATIR approach does not require databases to be pre-processed by encryption or be periodically shuffled.

Compared with the best solution known so far, the number of replicated servers required by ATIR is reduced to be optimal in the sense that no additional servers are required to satisfy the privacy property of ATIR. To tolerate any collusion of up to t servers and up to f faulty (crashed or malicious) servers in a system with k replicated servers, ATIR ensures the privacy and correctness properties, provided that $k \geq t + f + 1$ where $t \geq 1$ and $f \leq t$. The best solution (i.e. [BS02]) with an $O(n^{\frac{1}{\lfloor (k-1)/3t \rfloor}})$ communication complexity requires k ($\geq 3t + 1$) replicated servers to cope

with the same situation. In contrast, ATIR uses fewer servers with a much improved communication cost, $O(n^{\frac{1}{2}})$ (where n is the size of a database managed by a server).

The usefulness and practicability of ATIR is proved by practical implementation on realistic databases with good performance. The implemented systems are purely based on software. Neither secure hardware nor trusted third parties are needed to implement and deploy these systems. Experimental results show that the implemented systems also perform well in both normal and simulated faulty circumstances. The PIR and ATIR implementations build upon widely used database technologies (MySQL). No modification of the underlying database servers is required to support the deployment of the services.

Moreover, the experimental results show that the Total Time for Processing (TTP) of using PIR and ATIR services are both well under half a second for calculating over data sets of a size ranging from 100KB to 1MB in normal circumstances. The ATIR services maintain the *same* level of performance even in the presence of the simulated attacks.

1.4 Thesis Organisation

The remainder of this thesis is organised as follows.

Chapter 2 presents a comprehensive survey on the following topics. The basic concepts and terminologies used by this thesis are presented first. A classification of *attack models* and a discussion on the relationship between these models are then followed. As a first line of defence against attacks, *privacy protection techniques* are categorised and their principles are described. We then present a survey of the PIR model, schemes and extensions which is followed by a review and discussion of *attack tolerant techniques* used in defending against active attacks. A review of *common assumptions* used by attack-tolerant approaches in designing and deploying distributed systems is given.

Chapter 3 presents the system model, and construction of ATIR schemes. In particular, we show the link among three basic strategies used in ATIR schemes, which are privacy protection, error detection and attack tolerance. We analyse the intrinsic difficulties of using conventional error detection techniques in the settings of ATIR. We also present two ATIR schemes: one deterministic and one

probabilistic for ensuring information theoretic privacy protection and service correctness even in the presence of malicious attacks. Both schemes are replication-based. This chapter also offers a thorough comparison between ATIR and related PIR schemes.

Chapter 4 describes the architecture of ATIR systems and the realisation of ATIR schemes. Several important design and implementation issues are discussed. In particular, we describe how to extend the database model for performing ATIR computations.

Chapter 5 presents experimental results of this thesis. This chapter is organised into three parts. At first, we develop an analytic model for the performance of the ATIR service implemented. The model is used to validate the performance measurements of the experiments conducted. We then examine the impacts of various major parameters on the total processing time of an ATIR service in normal circumstances. The performance comparison between a probabilistic ATIR and a deterministic ATIR is then presented. Finally, we investigate the performance of an ATIR service in the presence of various simulated (crashed and malicious) faults.

Chapter 6 recaps the main contributions made by this thesis, discusses the future work, and concludes the thesis.

Chapter 2 Background: Coping with Attacks

This chapter reviews the state-of-the-art research efforts for coping with attacks in distributed systems. When attacks against computer systems are discussed, we should clearly state the following: i) types of systems being considered; ii) assumptions about system design, implementation, and deployment; iii) types of attacks that are defended. The first three sections of this chapter examine these issues in turn. In Section 2.1 we describe a general model for distributed systems and a client/server system model. In Section 2.2 we present and discuss two attack models: a passive and an active attack model. In Section 2.3 we review privacy protection techniques for defending against passive attacks and in Section 2.4 we present a survey on PIR research. In section 2.5, we focus on attack-tolerant techniques and systems for coping with active attacks.

2.1 Basic Concepts and Terminologies

This section describes a conceptual system model which will be used throughout this thesis. We will then emphasize the client/server design paradigm, a widely used structuring technique for designing distributed systems.

2.1.1 Systems, Services, and Components

A system is an entity composed of multiple interacting components which are under the control of a system design to deliver a service to other entities, i.e. other systems [PS01]. The other entities make up the environment for the system considered. A system *design* can be viewed as a special component of the system which not only defines the interactions among the components in the system but the interactions between the system and the environment that the system services [Xu04].

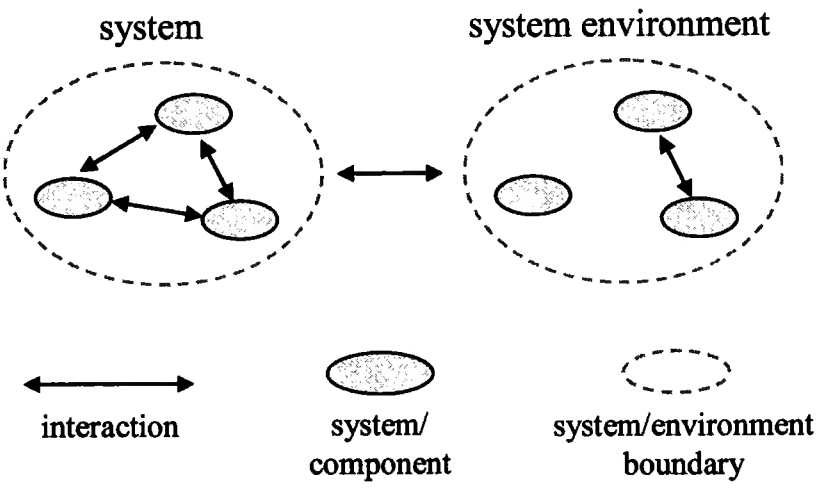


Figure 2-1 A Generalised Conceptual System Model (Adapted from [Xu04])

Figure 2-1 depicts a generalised conceptual model of a system and the system environment it services. From the diagram, we can see that the systems in the environment are loosely coupled since they may or may not interact with each other.

The system model is recursive in nature. The components in the system can be decomposed into sub-components. These sub-components in turn can be further decomposed, and so on. The decomposition stops when the component is atomic (i.e. not divisible). The divisibility depends on the level of abstraction needed [HW92].

2.1.2 The Client/Server System Model

Distributed systems usually are structured into one of the following four design paradigms: client/server, remote evaluation, code on demand, and mobile agent [FPV98]. It is important to note that design paradigms are independent of any particular programming languages, message transportation mechanisms and implementation technologies. The principle of our thesis applies to all of these paradigms. However, for simplicity of presentation, the client/server design paradigm will be emphasized.

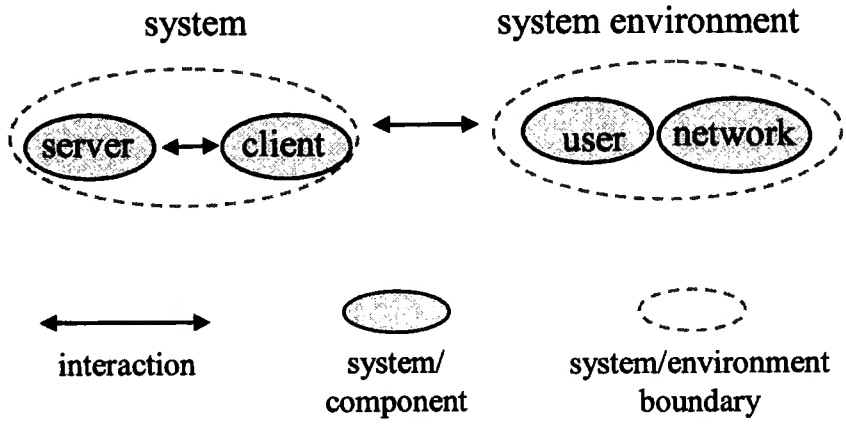


Figure 2-2 A System Model of a Client/Server Distributed System

As illustrated in Figure 2-2, a client/server distributed system is structured into two components: a client and a server. The client and the server interact with each other following a pre-defined protocol specified in the system design. The client and the server often run on separate processing nodes in a communication network. The server manages resources and defines operations that are exported to the client. The client sends a *request* message to invoke the operations provided by the server and receives a *reply* message in the opposite direction. The system can interact with the system (execution) environment in at least two ways. First, the system delivers a service to the environment by taking *inputs* from *a system user* and returning *outputs* back. The user is a special system in the environment that the system considered aims to service. Second, the interactions between the client and the server are built on top of a communication network which is also a part of the system environment. In this case, there is no interaction between the user and the network.

2.1.3 Security Failures

When being deployed in an error-prone environment, the system is often required to be dependable. A dependable system continues to deliver the intended service despite the occurrence of faults. Dependability is defined as the property of a computer system such that reliance can justifiably be placed on the service the system delivers [Lap92]. A dependable system needs to ensure that the service is available, reliable, safe, and secure. When a service delivered by a system derivates

from what it intends for (e.g. from the design specification), this event is defined as a system *failure* [Lap95]. An *error* is that part of the system state liable to lead to the failure [Lap92]. A *fault* is a hypothesised cause of an error [Lap92].

This dissertation focuses on security failures. Security is dependability with respect to the prevention of unauthorised access and/or handling of information [Lap92]. Security consists of three properties: confidentiality, integrity, and availability. A security failure occurs when one or more of these properties are violated. A security error is that part of the system state leading to the security failure. A security fault is the hypothesised cause of a security error. A security failure may be neither detectable, nor observable. For example, when the confidentiality of a piece of information is breached, a security failure occurs. However, due to the passive nature of information leakage, such a fault may not manifest itself as a detectable/observable failure.

2.2 Attack Models

We now examine how security properties of a service can be violated and the effects of such violations to the service from a security perspective. Security violations are often caused by attacks. An *attack* is defined as an intentional fault aiming to violate security properties of a service. Therefore, an attack is a type of fault. Consequently, attack models are a subset of fault models. Attack models are often used to categorise the aspects (e.g. types and assumptions) of attacks considered in a system design, implementation and deployment.

2.2.1 Classification of Attacks

The degree of security violations that may be caused by attackers is often used to characterise the power of attackers. As a result of the characterisation, system-level attacks are classified following two types of attack model:

- *Passive attack model*: passive attacks only involve monitoring, intercepting, or eavesdropping on information. Passive attacks often lead to the violation of the confidentiality property, or traffic analysis attacks.
- *Active attack model*: active attacks often build on passive attacks and involve intercepting, modifying, and probably fabricating information. Active attacks

often lead to the violation of all security properties (i.e. confidentiality, integrity and availability).

In real world, attackers often rely on the information gained through passive attacks to launch active attacks. Successive active attacks can also involve information disclosure, hence, causing the violation of confidentiality property. Therefore, the active attack model subsumes the passive one.

Classified by their effect (i.e. severity) on the components of the system, active attacks can lead to two types of failures: benign failures and malicious (also referred as Byzantine) failures [CDK01 pp. 56, Mul93 pp. 100]. Examples of benign failures include fail-stop, crash, and omission failures. Examples of malicious failures include arbitrary execution of a predefined protocol and/or malicious corruption of databases. The component that exhibits failures is described as a faulty component.

The purpose of having two attack models is to simplify the design of security protocols and to make realistic assumptions against attacks. Protocols with weak assumptions are often too expensive to be deployed. Therefore protocols design typically relies on certain assumptions [And01]. Assumptions are a double-edged sword. Without making assumptions, it is impossible to derive effective and useful protocols. Protocols can be made simple with certain assumptions. However, by invalidating assumptions of a system, the system will fail easily. The key issue is what are the valid assumptions, and under what circumstances.

2.2.2 Relations between Attack Models

Active attacks are often built upon findings through passive attacks because information collected during passive attacks is useful for conducting active attacks in the later stage. Both attack models apply to individual components, interactions among components, and a system as a whole. Consider a hospital database system which provides information about patients to authorised doctors. A doctor uses the client software installed on his desktop to access the database server. There is a communication network connecting the client and the server. In its simplest form, the system is composed of two components: the client software and the database server. If the operator only steals patients' information, the operator only conducts a passive attack which targets an individual component. If someone only manages to eavesdrop on the communication network, this is an attack against the interaction

between the client and the server. The system administrator can easily launch attacks against the entire system.

From a system's point of view, attackers can be classified into two categories: outsiders and insiders. In this thesis, we only consider closed systems. A closed system is defined as a system which has a clearly defined system boundary. According to the MAFTIA [PS01] report, an outsider is defined as "a person who has no privilege, i.e. no rights to any object in the system" whereas an insider is "any individual who has some privileges, i.e. some rights on objects in the system". Attacks launched by insiders are insider attacks whereas attacks launched by outsiders are outsider attacks.

2.2.3 Attacks on Systems and Cryptanalysis

There are differences between attacks on systems and those on cryptographic algorithms. Attacks on systems are the topic we have discussed in this section and are the main concern of this thesis. Cryptanalysis is the science and art of finding the weakness in cryptographic algorithms. The aims of attackers, targets and methods of attacks are different. Although a successful attack on cryptographic algorithms can lead to successful attacks on systems (i.e. those use these algorithms), current research tends to treat them separately. Cryptographic algorithms are building blocks of secure computer systems and often are assumed to be secure. Therefore, there is a need to clarify the common security assumptions made when designing and implementing computer systems.

A successful attack often leads to information disclosure. Considerable research has been done on understanding the effects and deriving effective countermeasures against information disclosure caused by the leakage of confidential information. The next section focuses on the concepts and approaches of information disclosure caused by the violation of personal private information.

2.3 Privacy Protection: Defending against Passive Attacks

The notion of "privacy" is central to many arguments about defending against attacks. In security, this term is often used inconsistently and/or loosely defined. For example, in [And01], Anderson defines the term privacy as follows:

> *"Privacy is the ability and/or right to protect your personal secrets; it extends to the ability and/or right to prevent invasions of your personal space (the exact definition varies quite sharply from one country to another). Privacy can extend to families but not to legal persons such as corporations."*
>
> ***Security Engineering***
> ***– A Guide to Building Dependable Distributed Systems***
>
> Ross Anderson

2.3.1 Defining Privacy

In this thesis, our definition of privacy is based on a restricted version of Anderson's definition. Privacy is defined as the ability/right to protect one's personal information, including secrets, from *unsanctioned* scrutiny. Examples of private information are identity, preference, location, and intention.

Privacy protection technologies aim at preventing unsanctioned information collection of individuals and protecting data that has been collected. This aim should be considered in a broad sense. Individuals should have the ownership to their private information: it should be up to them to decide who is allowed to access the information and how the information is distributed. That is, they should have the ability/right to control private information. However, it is often up to others to maintain the confidentiality of the information. For example, patient medical records are private information of the patient. However, it is often up to the hospital or health care personnel to keep them secret.

In the next section, we present privacy protection techniques which are categorised into the following approaches: encryption, anonymity, delegation, pseudonym, and policy negotiation.

2.3.2 Privacy through Encryption

Encryption is a method of information transformation in which the process transforms a piece of private information into an unintelligent form so that the exact

content is masked. The design of an inverse process (i.e., decryption) is important to the success of encryption. Decryption provides a way to recover scrambled information to its original form. Often, this reverse transformation is controlled by secret information (e.g., decryption keys) which is only accessible to authorised parties. Encryption often relies on number theoretic assumptions, such as the computational difficulty of factoring large prime numbers in cryptographic algorithms [RSA78].

By encrypting private information, its exact content can be hidden from attackers. However, once the information is decrypted, few technical measures can prevent further dissemination of the information. Performing encrypted computation is the approach for protecting the privacy of users without decrypting the content of computation.

There has been a large body of research on encrypted computation. Three methods are often used to perform encrypted computation: processing encrypted data [Fei85, ALN87, AFK87, BF90], computing with encrypted functions via privacy homomorphism [RAD78, ST98a, ST98b, Bre01], and processing obfuscated programs [WDHK01, ZGZ03]. Unlike normal computation/processing, encrypted computation does not need to be decrypted before processing.

This property is highly attractive because it means that users can utilize resources without information about their input data, functions, or program logic, being revealed. For certain applications, such as bioinformatics, proprietary and patented programs, this is an appealing feature because these applications often require a large amount of computational resources and have significant privacy concerns.

Currently, the first two methods (i.e. processing encrypted data and computing with encrypted functions) are restricted to limited types of functions and these functions often rely on computational assumptions, such as the difficulty of factoring large numbers. The major obstacle to adopting such methods is that there are not many functions that can be executed in an encrypted form. The general applicability of constructing and executing obfuscated programs, i.e. the third method, for real application also remains to be explored.

As a whole, all the research in this area largely focuses on demonstrating the theoretical feasibility of using these methods for solving restricted types of problems. Although there are potentially huge practical motivations and demand for methods that support encrypted computation, it remains to be seen how to extend and apply these methods for use in real applications. The state of the art results in this area still reside on a theoretical level. Little is known on how to apply them in real applications.

Moreover, all existing work using any of these methods assumes the passive attack model, where attackers will execute the predefined protocols correctly. This assumption becomes shaky when these methods are applied to solve privacy problems arising in large scale distributed computing, such as mobile code applications and Grid applications. The scale and dynamicity of these applications demand solutions which do not only protect privacy, but also guarantee the integrity of the computation.

2.3.3 Privacy through Anonymity

In order to protect the identities and locations of users, anonymity techniques involve the use of a large number of intermediate nodes (e.g. computers) to cooperate together to perform an operation. When a user sends a request, the request is often routed through these intermediate nodes. It is therefore difficult for an observer to determine the exact information (e.g. the sending location) of the request.

For example, by routing an email through a large number of intermediated nodes before sending it to the receiver, the identity and location of the sender can be hidden. Anyone in the participant set can be the one who sends the request and the initial origin of the request. Initiated by Chaum in 1981 [Cha81], there has been a large body of work (e.g. [Cha88, Wai89, RR98, MC00]) in this area. This work can be classified into two categories: anonymous remailers and anonymous web browsing. Anonymous remailers (e.g. MIX networks [Cha81, Cha88] and MIXMASTER [MC00]) aim to protect the identity and original location of email senders. Anonymous web browsing techniques protect the identity and/or location of web users and examples of such are CROWS [RR98], onion routing [GRS99], and Web MIXes [BFK00]. The research in this area is relatively well developed and systems (e.g. Zero-Knowledge System's Freedom Network [BSG00]) with varying degrees of

privacy properties and involvement from commercial companies have been developed.

In theory, the larger the population is, the harder it becomes to identify and pin-point the identity and exact location of the originator. Usually, the provision of anonymity-based techniques heavily relies on the availability of a large number of trusted intermediaries. These techniques are also coupled with encryption to further protect other private information of users, such as authorships and censorship. Recently, research has been begun to look at the privacy protection issues in the presence of certain types of attacks on anonymous networks (e.g. [Dan03]).

2.3.4 Privacy through Delegation

Some examples of this approach are Anonymizer [Ano04] and Rewebber [Rew04]. These web sites act as a "proxy server" to request senders. Using these websites, the identity, IP address, and other private information of a requestor can be hidden from web site owners. The retrieved pages are routed through these servers back to senders. One distinct feature about Rewebber.com is that it masks the location of documents requested from the request sender. This approach can protect the privacy of the document publisher. Both techniques rely on the trustworthiness of proxy servers to provide privacy protection.

A special form of privacy protection through delegation is the use of secure hardware. In this case, privacy protection is delegated to a maintainer or operator of a piece of trusted hardware which is assumed to be tamper-resistant (i.e. no secrets will be leaked even if the hardware itself has been damaged). Secure co-processors are often used to store private information and/or perform security critical operations. There remain several problems with the use of secure co-processors (for details, readers are referred to Section 2.5.5). Also, the memory and processing capability of secure hardware is limited, which means that they may not be applicable for computationally intensive applications.

2.3.5 Privacy through Pseudonym

Identity privacy has become one of the major security issues of Internet applications because of the rapidly emerging threat of identity theft [DOJ00]. To combat identity theft and reduce the damaging consequences of misusing personal information,

pseudonym techniques have been used for depersonalise uniquely identifiable information from transactions. To hide the exact information about a user, pseudonym systems replace the personal information with randomly generated pseudo-information.

Without using a user's real identity, the design and implementation of authentication and authorisation services for pseudonym systems become a challenge. Currently, there are two basic approaches to address this problem. Both do rely on a universally trusted authority to implement these services. For example, this is exactly the strategy adopted by the IBM's IDEMIX system [CH02]. Alternatively, one can employ a distributed solution, such as the one used by the Shibboleth framework [CE02]. Shibboleth, by itself, does not tie application developers to any specific authentication mechanisms. Authentication in Shibboleth relies on the existing security infrastructure provided by individual participating organisations. After a user has been authenticated by its origin organisation, depersonalised information is sent to a foreign organisation to authorise the user to certain resources. In this sense, Shibboleth provides a privacy-preserving authorisation to users.

2.3.6 Privacy through Negotiation

There are other approaches for privacy protection, such as using privacy policy negotiation mechanisms. The focus of existing work aims to automate the policy checking and negotiation processes. The success of these techniques often depends on the "good faith" of service providers. In this approach, a user can specify their privacy preference and negotiate with servers before a transaction begins. Currently, the work in this area aims to provide standard ways for achieving policy negotiations and examples using this approach include P3P [P3P01] and Appel [Appel02].

2.3.7 Privacy through Secret Sharing

Secret Sharing [Sha79, Bla79] is a classical security approach for ensuring the confidentiality and availability of private information, such as long term signing keys and personal private keys. With the use of secret sharing techniques, a secret, in its simplest form - a number - is split into randomly generated shares which are distributed to mutually independent participants. A threshold number of shares can be used to reconstruct the original secret. The loss of secrets usually causes

catastrophic consequence. For example, the loss of signing keys of a Certification Authority invalidates all the certificates issued whereas the compromise of the latter leads to the potential for invasion of private/confidential communication or data. For example, the COCA system [ZSR02] and SINTRA system [CP02] use secret sharing techniques to protect service private key(s).

Secret sharing has also been used to protect the privacy of authors in the application of online publications. For example, Publius [WRC00] uses secret sharing to provide privacy protection for authors when they publish documents through the Publius system.

2.4 Private Information Retrieval (PIR): Model, Schemes, and Extensions

This section presents the background on the PIR problem [CGKS95] and a survey of PIR schemes.

2.4.1 The PIR Problem

Consider a database query scenario [CGKS95]. An investor queries a stock share database for the value of a stock share but s/he doesn't want to disclose the identity of the specific stock of interest. This is the classical motivating example of the PIR problem which is concerned with querying databases privately - without revealing the identity (i.e. intention) of the specific data item of interest to the database owner.

2.4.2 The PIR Model

The PIR problem is often considered in the following model. The database is modelled as a binary string $x = x_1 \ldots x_n$, where each bit represents a data item in the database, the suffix is the index of each data item, and n is the size of the database. The user knows an index i of a data item x_i and is interested in getting x_i. With a single server, in order to protect the privacy of the user without leaking any information about the identity to the database server, the user has to download the entire database and perform the query locally [CGKS95]. Essentially, this result states that it is impossible to obtain unconditional privacy protection with the use of a single server. The communication complexity of this solution is $O(n)$. When a PIR scheme has communication complexity $O(n)$, it is a trivial solution [CGKS95]. The

communication overhead of this solution prohibits its practical usage. Also, downloading is simply not a practical choice in some circumstances (e.g. patients' data or paid services) due to the proprietary nature of data and/or cost issues. Therefore, reducing communication complexity becomes one of the major issues in the PIR research, that is, achieving non-trivial communication complexity becomes one of the main goals in current PIR research.

PIR schemes are concerned about hiding the user's intention (i.e. the index i) from the server. PIR schemes should have less than $O(n)$ communication complexity. From a practical point of view, the PIR problem has a wide range of applications in the traditional security and privacy sensitive domains, such as banking, insurance, and e-government. In such domains, a vast amount of research has been done in protecting servers' privacy whilst users' privacy protection problems remain largely open.

All PIR schemes consider a passive server attack model where the attacker who controls the server (s) can only passively observe the processing on the server(s) *and* messages exchanged between the user and the server (s). In the passive attack model, PIR schemes need to consider only the privacy of users. However, as indicated in Section 2.2, the passive attack model is a restricted version of an active attack model, where an attacker is assumed to be only interested in finding out the identity of the data item but follows the protocol correctly.

2.4.3 Information Theoretic PIR Schemes

A high level overview of a PIR scheme is described as follows to explain the principle of PIR schemes. A PIR scheme consists of three algorithms: a query algorithm, an answer algorithm, and a reconstruction algorithm. At first, the index of the data item - an integer - is first transformed into a sequence of numbers following the query algorithm by the user. The sequence is then sent to the server and used as inputs for the answer algorithm. The answer algorithm processes through each record in the entire database and produces an answer. Based on the answer returned, the user executes the reconstruction algorithm to produce a result. Since every record in the database is involved in the server side processing, the server has no way to figure out the information about the intended data item. Therefore, in principle, all PIR

schemes are based on the privacy through anonymity approach for protecting users' privacy.

In order to achieve non-trivial communication complexity, two approaches have been derived to achieve better communication complexity while still maintaining privacy protection for users. The first approach is to relax the single server requirement to the use of multiple servers. Using this approach, the PIR research has been explored along two directions. One can either use multiple replicated servers (i.e. replicas) to store identical copies of data. This is the approach used by [CGKS95, Amb97, BI01, BIKR02]). Also, auxiliary servers have been introduced to store randomised data before a PIR scheme begins. This is the approach used by [GGM98].

In the replication based PIR schemes, such as [CGKS95, IK99, BI01, BIKR02, Amb97], identical database servers are replicated on separate nodes of a distributed system. It is assumed that communication between the replicated servers is restricted. No more than t ($t \geq 1$) servers are allowed to communicate with each other so that the servers cannot collude together in trying to violate the user's privacy. Except in one case [CG97], all replication based PIR schemes provide privacy guarantee unconditionally. No information about the user' intention can be obtained by the servers.

In the replicated-server setting, Beimel et. al. in [BIKR02] construct several information theoretic PIR schemes with less than $O(n^{1/(2k-1)})$ communication complexity for $k \geq 3$. These are the best PIR schemes in a replication setting when this thesis is written. However, a small number of replicated servers is particular interesting because they place less demand on the deployment setting. At the time this thesis is written (i.e. September 2004), the minimum number of servers required by PIR schemes is two. The best communication complexity of two-server PIR schemes is $O(n^{1/3})$, which is achieved in [CGKS95, BIKR02].

2.4.4 Computational PIR Schemes

Alternatively, the second approach relaxes the privacy requirement from unconditionally secure to computationally secure. Hence, computational PIR (cPIR) schemes are constructed based on computational assumptions. With this approach, one can construct single-server PIR schemes to protect the users' privacy. Loosely speaking, unconditional privacy means the queries cannot provide any useful

information to computational unbounded servers with respect to the intention of the user. The implication of this approach is that the attacker who controls the server has to be assumed to be computationally bounded.

Chor and Gilboa [CG97] proposed the first cPIR scheme with the use of (at least) two non-communicating servers and this scheme assumes the existence of pseudo random number generators. The communication complexity of this scheme is $O(n^{\varepsilon})$ communication complexity for any positive real number $\varepsilon > 0$. The communication restriction and the use of multiple servers are essential in the first cPIR scheme since the collusion between servers can completely reveal the user's privacy.

In the same year, Kushilevitz and Ostrovsky [KO97] proposed the first single-server cPIR scheme based on Quadratic Residuosity Assumption [GM84]. The work first shows the possibility of getting rid of the multiple-server setting with the use of a well established cryptographic problem. The communication complexity of their scheme is also $O(n^{\varepsilon})$. Yamamura and Saito in [YS03] propose a single-server cPIR scheme of communication complexity $O(n^{c})$, where $c > 0$, based on the subgroup membership problem.

Subsequently, Cachin, Micali, and Stadler [CMS99] presented the first single-server cPIR scheme of polylogarithm communication complexity based on a somehow less well-known number theoretic assumption Φ–hiding assumption.

Recent studies show that some existing single-server PIR schemes already have close to optimal communication complexity [WW04]. As long as these problems are still hard to solve in a computational sense, the user's privacy is preserved. The relaxation of the privacy requirement in computational PIR schemes is an important step towards a single server PIR scheme which is more interesting in practice due to the replication setting of unconditionally secure PIR schemes.

2.4.5 PIR Extensions

The PIR model has been extended along several directions. Symmetric PIR schemes (SPIR) [GIK+98] were proposed to protect both the privacy of database servers and users. SPIR schemes ensure that a curious user (even not correctly executing a SPIR scheme) cannot obtain extra information from the database except the single bit of the data item originally intended for.

Gertner et. al. in [GGM98] propose a random server model for achieving unconditional privacy protection without using replication. Secret sharing schemes [Sha79] are used to generate the contents of a number of auxiliary databases randomly based on the content of the original database. Even an attacker can observe the processing on all auxiliary databases; no information about the user's privacy will be revealed. However, the attacker is assumed to not be able to access the original database. To complete a PIR scheme, the user needs to access all databases, including the original and auxiliary ones.

The PIR model assumes that the user knows the index of the data item of interest. In practice, the index information is not always readily available. A database is typically presented by keywords. Chor et. al. [CGN98] extends the PIR model with added keyword search capability. In their schemes, the database owner is required to insert a sequence of keywords (binary strings) into a data structure which supports the search function. The user and the database collaboratively finish a sequence of computations in a fixed number of rounds. The result of each computation determines the address of the keyword to be fetched in the next round. The address is used to perform a PIR scheme to fetch the data item accordingly. No modification of the database and underlying data structures is required to support these schemes.

These extensions are based on the standard binary bit model and a passive server attack model. Consequently, all database servers are assumed to execute protocols correctly. They all add a constant level of complexities (both communication and computation) to the standard PIR schemes. However, the levels of complexities are added at different stages of PIR processing. The random server model based PIR schemes require an introduction of a number of additional auxiliary servers with restricted communication among them before the PIR processing begins.

The keyword-based PIR schemes need a modification of the original database to support the search facility and a constant number of execution rounds is needed to complete a keyword-based PIR scheme. Finally, the databases in SPIR schemes do not need to be pre-processed. However, the communication and computation complexity of information-theoretic SPIR schemes are much better than those of computational SPIR schemes [Mal00 pp. 111]. But computational SPIR schemes are more attractive in practice due to their single server setting.

All these extensions introduce complexities on top of those of standard PIR schemes, such as those presented in [CGKS95]. Hence, they may be too costly to be implemented.

2.4.6 PIR and Active Attacks

The standard PIR model assumes a passive server attack model. In the multiple-server PIR schemes, only up to a threshold number of servers are allowed to collude together to perform passive attacks. In the single-server PIR schemes, the server is also restricted to a passive attack model. In reality, once an attacker occupies a server or a threshold number of servers, active attacks may be conducted. With the use of an active server attack model, PIR schemes need to consider not only the privacy of users but also the correctness of results.

The PIR problem has been extended to deal with two types of effects on the servers caused by active attacks: crash failures and malicious (Byzantine) failures. Beimel and Stahl in [BS02] refer to the PIR schemes in these failure models as robust PIR schemes. Yang, Xu, and Bennett in [YXB02a, YXB02b, YXB03] also consider the same problem under these failure models, and the PIR schemes developed are called attack-tolerant (fault-tolerant) information retrieval schemes. Due to the use of the active attack model, these PIR scheme are required to guarantee the privacy of users and ensure the availability and correctness of results even in the presence of failures.

2.4.7 Hardware-based PIR Schemes

In [SS00, SS01], two researchers Smith and Safford from the IBM Watson research centre initiate hardware-based PIR research and propose several PIR schemes based on secure co-processors. With the same goal as the research conducted in this thesis, hardware-based PIR also aims at practical implementation and deployment of PIR schemes. This section only provides an overview of hardware-based PIR and a comparison between these schemes and the ATIR schemes presented in this thesis appear in Section 3.4.2.

A typical architecture of hardware-based PIR schemes is depicted in Figure 2-3, which is adapted from the system architecture presented in [IS03]. In order to hide the identity of the intended data item, a host is required to execute the proposed PIR schemes inside secure co-processors which are responsible for two tasks: shuffling

the database regularly and performing retrieval operations. In some other implementations [Aso04], all the tasks are assumed to be performed within one single co-processor.

There are conceptually two types of secure co-processors: a shuffler and a PIR server. A shuffler periodically shuffles the records in the database. The shuffled records are subsequently encrypted and put back to the database by the shuffler. During shuffling, no PIR operations are allowed to be performed on the database.

The PIR server accepts users' queries, retrieves information from the database, and returns encrypted results back to the client. Each record in the database needs to be iterated through. (Otherwise, it is clear that the intended data item is not those "untouched" records.)

Encryption keys have to be shared among each pair of secure co-processors and between the client and the PIR server. The shadowed boxes in the figure depict the places where data is encrypted. The dotted boxes represent the places that encryption/decryption operations take place. The information exchanged between the client and the secure co-processors are encrypted so that the server cannot obtain the information exchanged. The users' privacy is preserved given that the secure co-processor(s) is tamper-resistant.

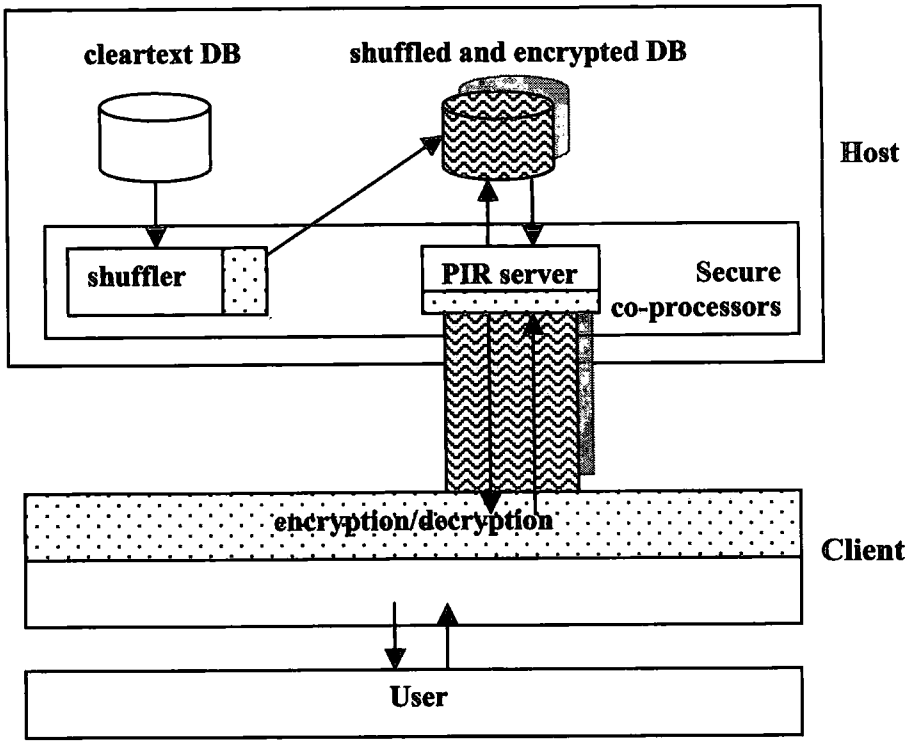


Figure 2-3 An Architectural Overview of Hardware-based PIR Schemes

2.5 Attack Tolerance: Thwarting Active Attacks

Prevention-based approaches, such as the privacy protection techniques discussed in Section 2.3 and 2.4, are not sufficient to solve the problems caused by attacks. In most real world scenarios, the information collected through passive attacks is often used to launch active attacks against a system [Shi04]. To effectively deal with attacks, another approach – attack tolerance is therefore introduced and applied to system design to prevent the security properties of a system being violated.

This section is organised as follows. In Section 2.5.1, the term attack tolerance is defined. From Section 2.5.2 to Section 2.5.4, we review three commonly used approaches for achieving attack tolerance in designing and deploying distributed systems. In reality, the effectiveness of these approaches heavily relies on the validity of the assumptions these approaches make about the deployment environment. Hence, in Section 2.5.5, a number of commonly used assumptions made by attack tolerant systems are examined and critically discussed. The aim is to highlight the risks of making invalid assumptions and to understand the limitations of the existing attack tolerance approaches.

2.5.1 Defining Attack Tolerance

Attack tolerance is a systematic approach for ensuring the delivery of correct services in spite of attacks. Specifically, the aim of using attack tolerance approaches is to guarantee the security properties of a system despite the occurrences of attacks.

The aim of an attack-tolerant system is to prevent security faults within some components in the system from manifesting themselves as a system security failure. Particular emphasis within attack tolerance is placed on the measures of dealing with attacks. As indicated in the attack models presented in Section 2.2, passive attacks violate the confidentiality property. Hence, they lead to information disclosure. In contrast, active attacks may violate all security properties and render service disruptions (e.g. unavailability of correct results). The design goal of such systems is to provide continued availability and graceful degradation of system services in the presence of attacks, maximising the residual capacity available to legitimate users. Attack tolerance research represents an effort in combining and unifying existing security and fault tolerance techniques.

2.5.2 Attack Tolerance through Secret Sharing

Apart from being used to defend against passive attacks, Secret Sharing techniques [Sha79, Bla79] have been widely used in the design of attack-tolerant systems to protect secret information, such as long-term encryption keys and private data storage, and to ensure service availability in the presence of attacks.

The security and availability of the information is assured so long as the number of corrupted parties does not exceed a predefined threshold. Secret sharing is a cryptographic primitive having been widely used to protect the security and availability of secret information (see, for example, [Cac03, ZSR02, CP02, CS03]).

Secret sharing consists of three parts: secret splitting, secret distribution, and secret reconstruction. In its simplest form, a secret sharing scheme is a threshold scheme by which a secret is randomly split into mutually independent shares which can be distributed to mutually suspicious participants. A standard setting of secret sharing is to employ a trusted dealer who performs the secret splitting.

Secure channels are required by secret sharing schemes to ensure that each share remains secret to other participants during transit. Since participants do not trust each other, shares are kept secret until secret reconstruction. When the shares are distributed, the original secret will be destroyed. Since each share is independent

from each other and the splitting is random, each individual share reveals no information about other shares or the secret. That is, none of the participants has any knowledge of what other people have.

There is a reverse process - secret reconstruction. This process is usually done by another trusted agent who pulls all the shares from an authorised set of participants for recovering the secret.

Most of existing secret sharing schemes rely on a synchronous communication model [Mul03]. Due to the increasing popularity of Internet based attacks, recent research in this area focuses on extending the previous techniques in an *asynchronous* setting, such as the Internet. Secret sharing techniques, in particular verifiable secret sharing techniques [Fel87], have been extended to an asynchronous setting to deal with active attacks in an asynchronous environment.

The COCA system [ZRS02], an online certification authority designed and developed in the University of Cornell, first propose the Asynchronous Proactive Secret Sharing (APSS) protocol. The aim of the COCA system is to protect the confidentiality of the signing key of an online certification authority and ensure the service availability even in the presence of active attacks to an asynchronous system.

The Delta-4 project developed a distributed fault tolerance framework using secret sharing protocol to store sensitive information into non-trusted sites/servers for open distributed systems [DBF91]. The Fragmentation-Redundancy-Scattering (FRS) technique was developed for tolerating both accidental and intentional faults [FDR95].

2.5.3 Attack Tolerance through Threshold Cryptography

Threshold cryptography is a non-trivial extension of secret sharing by allowing the cooperation of a number of mutually suspicious participants to complete a task (e.g. signature signing) *without* revealing the secret (e.g. a signing key). For example, in a threshold signature scheme, a secret signing key is shared among n participants and each of them can generate shares of signatures given a message. A valid digital signature can be generated from any $t + 1$ valid signature shares and its validity can be verified by a publicly known verification key.

The Stanford's Intrusion Tolerance via Threshold Cryptography project provided tools and infrastructure for building intrusion tolerance applications, specifically a

Certificate Authority based on distributed RSA signatures, embedded into a web server [WMB99].

The IDA algorithm [Rab89] has been applied to protect document integrity in the e-Vault system [ICG+98], an online, distributed data repository developed at IBM Watson Research. Shared distributed digital signatures are generated by the e-Vault system and given to users as a proof of proper storage of documents. The design goal of the e-Vault system is to guarantee the secure storage and retrieval of data in a distributed storage system.

In the EU MAFTIA project, the SINTRA system [CP02, CS03] is designed and implemented to protect critical online services such as DNS. SINTRA is based on a variant of threshold cryptography.

2.5.4 Attack Tolerance through Proactive Security

Traditional security mechanisms often assumed that all the security-critical components of a system must be secure throughout the lifetime of a system. This goal is extremely hard to achieve in real world applications. Instead, the proactive security techniques, as proposed in [Jar95, FGM+96, CGH+97, HJJ+97], are based on the assumption that an attacker can never compromise more than t components during a time period. Provided a sufficient number of critical components in the system are secure, the overall system security can be guaranteed even when some components have been compromised. Proactive security mechanisms consist of two operation stages: distribution of secrets, and periodic refreshment of secrets. For example, with the use of secret sharing, shares of a secret can be randomly generated and distributed to a set of components in a system. These shares can be refreshed periodically without the need to reconstruct the original secret. Notable examples of using proactive security include AT&T's Omega project [RFL+96], Byzantine Fault Tolerance (BFT) [CL99], and COCA [ZSR02].

In the AT&T's Omega project [RFL+96], the principle of proactive security was used to build a highly resilient and distributed key management service - the Rampart Toolkit - which can tolerate the arbitrary corruption of some of the servers by an attacker.

Castro and Liskov [CL99] present a practical Byzantine Fault Tolerance (BFT) algorithm in an asynchronous setting with the use of symmetric key cryptography.

BFS – an implementation of BFT on NFS performs very well if no failures occur. BFT heavily relies on proactive security to recover compromised servers. Periodic refreshment of secret keys invalid the information obtained by attackers.

The Cornell Online Certification Authority (COCA) [ZSR02] also uses the principle of proactive security to thwart against active attacks. All these systems use the proactive security mechanisms to recover compromised components.

The implementation of proactive security in all these systems relies on the availability of tamper-resistant hardware (e.g. secure co-processors) to store encryption and signing keys. All cryptographic keys are stored within the co-processors and all security sensitive operations are assumed to be performed by the hardware. Hence, even if a server is compromised, the security of the keys, such as the confidentiality of encryption keys and the integrity of verification keys, can still be maintained.

An alternative approach is suggested by Zhou [Zho01] to enable proactive security without the use of secure hardware. In this approach, trusted operators are employed to manually propagate security keys through secure offline channels. However, this approach seems even more restrictive than the secure-hardware approach due to the compulsory involvement of regular manual reconfiguration.

2.5.5 Commonly Used Assumptions

This section examines some common assumptions made by attack-tolerant systems, and aims to highlight the risks of making invalid assumptions and point out the limitations of existing attack-tolerant approaches.

The Soundness of Security Parameters

When a system incorporates cryptographic algorithms in its design, two assumptions are often considered. The first assumption is to assume the security parameters chosen for these protocols are sufficient, and that these protocols have been implemented properly to ensure the validity of their security properties. It is known that some cryptographic algorithms can become vulnerable due to badly chosen security parameters and/or poorly implemented algorithms [And01, Cop84]. This assumption is only justifiable when parameters are carefully chosen and algorithms are properly implemented.

The Use of Probabilistic Cryptographic Algorithms

The second assumption assumes cryptographic algorithms employed in the system design are perfectly secure, which is referred to as secure cryptographic algorithm assumption. This is probably the most commonly used security assumption with which numerous distributed systems are designed. By perfectly secure, it means that cryptographic algorithms ensure security properties with a probability one. In reality, these algorithms are often probabilistic because most of them are based on number theoretic assumptions which are probabilistic in nature. For example, all systems employing digital signatures rely on this assumption to check message integrity. Corrupted messages can only be identified within a certain probability, and there is a probability that a corrupted message cannot be identified. Therefore, this assumption is also referred to as a perfect failure detector assumption, because probabilistic cryptographic algorithms are treated as deterministic. Some example systems using this assumption include COCA [ZSR02], BFT [CL99], and SINTRA [CP02].

However, the soundness of this assumption depends on the validity of computational assumptions made about signature algorithms. When an assumption of such becomes no longer valid, the corresponding cryptographic algorithm will not be secure. Hence, it is necessary to know the risk of relying on this assumption.

Risks of Using the Secure Cryptographic Algorithm Assumption

The rapid development of cryptanalysis often has great implications of invalidating previously known-to-be sound cryptographic assumptions. Two examples of commonly used cryptographic assumptions are the difficulty of factorising integers and the collision-resistant property of hash functions. The popular RSA cryptosystem [RSA78] is based on the former whereas MD5 and SHA hash functions are based on the latter.

In the past, it was believed that the probability of producing collided hashes was negligible and it took millions of years to break the computational assumptions if parameters were chosen properly. But now, Wang et. al. [WLF+04] have demonstrated that breaking MD5 takes just a matter of hours using a standard laptop computer in the CRYPTO'04 conference! According to a report on CRYPTO'04 written by Edward W. Felten,

> *"Where does this leave us? MD5 is fatally wounded; its use will be phased out. SHA-1 is still alive but the vultures are circling. A gradual transition away from SHA-1 will now start."*

This latest breakthrough in cryptanalysis of hash functions certainly shakes the foundation of collision-resistant property of hash functions [McC04] because hash functions are widely used in numerous practical security protocols. For example, the Apache web server products and Sun Microsystem's Solaris products use MD5 to verify the integrity of their software distributions.

Hash functions are the foundation of many areas of modern cryptography, such as digital signatures, encryption, authentication, and file integrity checking. For example, digital signatures heavily rely on hash functions to produce collision-free digests of different messages. Nearly all secure and fault tolerant systems rely on digital signatures to detect message corruption during transmission, or to prevent corrupted parties from modifying messages sent by correct parties in group communication protocols. The importance of the cryptanalysis result means that attackers may be able to fabricate a piece of corrupted message to be validated as a correct one and hence the authenticity of the message may no longer be guaranteed. Therefore, the second assumption is not always sound and it should be often revisited when designing computer systems.

Secure Bootstrapping Assumption

Assumptions are also made in the implementation stage of a system. For example, the implementation of security services often relies upon the off band (i.e. cruiser service) distribution of security keys (e.g. encryption and signing keys). When a system is setup, a secure bootstrapping process is often assumed to exist to distribute a set of secret information (e.g. private keys) to each participant. Examples include COCA [ZSR02], BFT [Cas01], and SINTRA [CP02]. All these systems assume a

secure bootstrapping process to perform initialisation. The consequence of this assumption is that the system assumes a static membership among its participants. Extra services are needed to support dynamic addition and removal of members. Currently, there is no known way to do that without reinitialising the system [Cac04].

Secure Hardware Assumption

The secure bootstrapping assumption sometimes is replaced by the secure hardware assumption. Secure hardware (e.g. tamper-resistant secure co-processors) is used to process critical security operations and store cryptographic keys (e.g. long term keys). This assumption is often required for the purpose of proactive security in order to defend against malicious attackers. After rebooting the system from scratch, the confidentiality and integrity of the secret information are still preserved because of secure hardware.

However, a system still needs a way to distribute secrete information to secure hardware when the system initialises. This is often achieved with the employment of trusted human operators who manually set up the system. Once initial keys are distributed securely, further keys can be generated via various authentication and key agreement protocols. There remain many possible attacks which may invalidate the security of tamper-resistant hardware when they are used in practice (for a comprehensive description of these attacks against secure co-processors, see [And01]).

2.6 Summary

The quest for dependable services was, is, and will continue to be a major focal point in designing, implementing, and deploying modern distributed systems. However, as the scale of a distributed system increases, so does the number of components in the system, therefore, so does the probability that some components will fail [Sch93]. Due to the ever rising number of malicious attacks against computer systems, delivering dependable services has never been so challenging. To understand the new challenges and limitations of existing approaches, this chapter takes a snapshot of the state-of-the-art techniques and systems for coping with attacks.

To set the context of the thesis, we first present a generalised conceptual system model for distributed systems. The model is abstract enough to capture the essence of

a distributed system despite its size and dynamicity. This is followed by an in-depth discussion on two types of attack model: a passive attack model and an active attack model. To defend against passive attacks, a prevention-based approach - privacy protection - can be used.

However, in most realistic execution environments, active attacks will be encountered. Due to the size, complexity, and dynamicity of modern distributed systems, a prevention-based approach is not always effective and is often costly for keeping attackers outside the system. Hence, the attack tolerance design paradigm is needed to cope with the situation by leveraging a combined use of secure and fault tolerant techniques.

Hence, the last part of this chapter reviews the common attack-tolerant techniques and critically examines some commonly used assumptions by attack tolerant systems.

Chapter 3 Attack-Tolerant Information Retrieval (ATIR)

This chapter presents two replication-based Attack Tolerant Information Retrieval (ATIR) schemes which are the theoretical core of this thesis. In Section 3.1 we highlight the major considerations for constructing a practical ATIR scheme. In Section 3.2 we describe the system model that ATIR schemes aim to deploy and formally define ATIR schemes. In Section 3.3, the detailed algorithms of the ATIR schemes are described together with their communication complexity analysis and formal proofs. Section 3.4 compares ATIR and several relevant PIR schemes. Section 3.5 discusses the validity of ATIR assumptions, and compares ATIR with other secure and fault tolerant schemes.

3.1 Introduction

The core of the PIR problem is about privacy protection with non-trivial communication complexity. The use of multiple servers (e.g. replicated servers) is needed for any PIR schemes if information theoretic privacy is required [CGKS95]. In all existing multiple-server PIR schemes (e.g. [CGKS95, Amb97, IK99, GGM98, BI01, BIKR02, BIM00]), the availability of all servers is needed to ensure the correctness of a result. Hence, the standard PIR problem has to use passive attack models. Such a model basically assumes that all servers are available all the time and execute predefined protocols correctly. This assumption places significant restrictions on the types of attacks that an attacker may perform.

Once occupying a PIR server, an attacker may conduct active attacks against a PIR service. The types of active attacks may include a combination of the following: crashing servers, blocking communication links, sending wrong replies, or simply deviating from predefined protocols in an arbitrary way. Unlike traditional replies from servers, PIR replies are not straight answers (otherwise, a server knows the user's intention) and a reconstruction algorithm is needed to "recover" the actual

results. Hence, the consequence of successful attacks may not be easily detectable using traditional error detection techniques because attacks are usually assumed to occur during message transmission rather than server side processing.

The ATIR schemes, introduced in this chapter, aim to solve the PIR problem based on an active attack model. In these schemes, apart from reducing communication complexity, replication is also used as a means for dealing with active attacks. ATIR schemes have two main requirements. First, since an ATIR scheme is a generalisation of a PIR scheme (when there is no faulty server in an ATIR system), it should have a *non-trivial communication complexity* (i.e. less than $O(n)$, where n is the size of the database). A trivial solution (i.e. through database downloading) would be sufficient. In theory, the communication complexity of a replication-based PIR scheme can be reduced when more servers are used.

In practice, replication is always associated with deployment issues. The need for a large number of replicated servers often restricts the scalability of a system and limits the applicable application domains of a PIR scheme. Moreover, as the number of replicated servers increases, the actual number of bits exchanged over the network also increases, which, to some extend, contradicts with the original motivation of using replication. To a great extent, the practicability of ATIR schemes is inversely proportional to the number of servers required. Therefore, we are particularly interested in ATIR schemes which can provide non-trivial communication complexity with a *small number of servers*, which is the second requirement.

Before presenting the ATIR schemes, we introduce the preliminaries. Specifically, we describe the system model of these schemes and formally define ATIR schemes in the next section.

3.2 Preliminaries

3.2.1 Basic System Model

Consider a distributed system implementing a database query service. The system is comprised of one client and k replicated servers $S_1, \ldots, S_k$. There can be more clients but they invoke the service independently from each other. For simplicity of presentation, we assume there is only one client in the system. These components run on separated processing nodes connected by a communication network: there are k pairs of one-to-one independent communication channels between the client and

each server. The client and the servers communicate with each other by exchanging messages. The message from a client to a server is called a query whereas the message in the opposite direction is called an answer.

These channels are authenticated but are not assumed to be secure. That is, two communicating parties in this network are certain of the identities of each other. But neither message confidentiality nor integrity exchanged through these channels is assumed. The state of a channel is determined by the state of the server it attaches to. The possible states of a server shall be elaborated in Section 3.2.2.

The system assumes a synchronous model of computation, that is, there exist known bounds for both the execution speeds of the servers and message delays. Specifically, we assume that each message is received within δ time units after being sent.

Database Model

In normal circumstances, each server has an identical copy of a database. The database is modelled as a character string $x = x_1x_2...x_n$, where n is the number of records in the database. Representing a record in the database, each character is considered to be an integer taken from a certain integer set $\{0, 1, ..., X\}$, that is, $x_j \in \{0, 1, ..., X\}$, where $j = 1, ..., n$. The subscripts represent the index of the records in the database. For example, in extended ASCII encoding scheme [ISO8895], each character is associated with an integer taken from the set $\{0, 1, ..., 255\}$.

An Overview

Before going any further, we present an overview of how an ATIR scheme works. Suppose a user has an index i and is interested in obtaining the character x_i, where $i \in \{1, ..., n\}$. The user invokes the service through a client by giving i as an input and then awaiting a result *res*. The service should have two properties. First, it protects the privacy of the user (i.e. the intention) through keeping the input i secret from the servers. Second, it ensures the delivered result to be correct, that is, *res* is indeed the intended character.

Let r denote a set of local random integers generated by the client and L_r be the number of integers in the set. Based on i and r, the client produces k sets of *random and independent* numbers to form queries, using *k query algorithms* $Q_1, \ldots, Q_k$. The queries are denoted by $q_1, \ldots, q_k$. Let q be the query which has a maximum number of numbers and we denote this number be L_q. The randomness and independency of the queries ensure that the servers cannot derive useful information about the input. The queries are sent to the servers, respectively. Based on x and q_j, the server S_j produces an answer a_j by executing an *answer algorithm* A_j, where $j \in \{1, \ldots, k\}$. a_j is sent back to the client. Similarly, let a be the answer which has a maximum number of numbers and we denote this number be L_a.

Based on r, i, and some of the answers, the client repeatedly executes a *reconstruction algorithm* $\Re$ to produce results until a result is deemed to be valid. A result can be in either of the two states: valid or invalid. A valid result is in the set $\{0, 1, \ldots, X\}$. Otherwise, it is invalid. Valid results are further divided into two groups: correct results and incorrect results. The possibility of reconstructing valid but incorrect results is characterised by a parameter - undetected error rate ε. ε is the probability of the occurrence of valid but incorrect results. An *error detection function* is employed to distinguish valid results from the invalid ones. Together, the reconstruction algorithm and the error detection function constitute a *result verification algorithm* $\Im$ which verifies the correctness of the results and produces an output for the user.

Computation

All the computations involved in the above algorithms are performed over a finite field Z_p, where p is a prime number and $p \geq max[X/\varepsilon, k, X]$, where $\varepsilon > 0$. Combined with the database model, we also have $x \in \{0, 1, \ldots, X\}^n \subseteq Z_p^n$. As a finite prime field, Z_p can be denoted by the set $\{0, 1, \ldots, p - 1\}$. It is known that the operations over a finite field are closed, i.e. the results of addition, multiplication, subtraction, and division, are still elements of the finite field [LN83].

ATIR schemes are based on the arithmetic of finite fields. The client calculates the order of the field, i.e., p, based on the ε specified by the user and the valid range of database characters, i.e., X. p is determined before the query algorithm starts and is used by all the finite field computations in ATIR schemes. Since ε is specified by the

user, p can be determined on a case-by-case basis depending on the user's requirements. The client informs the servers about p along with the queries. For clarity of presentation, the algorithm for calculating p is deferred to Section 3.3.4 along with the presentation of an error detection function. Before that, we assume p has been chosen properly.

3.2.2 Servers and Attack Models

We consider an active attack model for our system. In the system, a server can be in one of the four possible states: correct, curious, faulty, and curious-and-faulty. A correct server follows predefined algorithms correctly and is not interested in finding out the identity of the item being retrieving. A curious server executes predefined algorithms correctly but attempts to violate the user's privacy. A faulty server exhibits failures by not following the algorithms. The answer returned from a faulty server is refereed to as corrupted. A curious-and-faulty server exhibits failures and attempts to violate the user's privacy. As already mentioned, the state of a communication link is determined by the state of the server it attaches to. For example, an attacker may attack a link through eavesdropping. But because the server it attaches to also is curious, we can simplify the presentation by treating the link and the server as one. In our system model, there are little differences between the effects caused attacks during transit and those caused by attacks during server side processing. For simplicity of presentation, we assume the communication links are reliable and secure hereafter. Therefore, we ignore the description of communication links in the remaining parts of the presentation when no confusion can be caused.

3.2.3 System Assumptions

In this system, three assumptions are made: 1) there are no more than t, where $t \geq 1$, curious servers who collude together to violate the privacy of the user; and 2) the number of faulty servers is bounded by f, where $k \geq t + f + 1$, $f \leq t$; 3) the client remains trusted throughout the lifetime of the system: the client does not collude with any servers and performs operations correctly. In Section 3.5, we shall discuss the issues of how to realise these assumptions and the validity of making these assumptions. These assumptions govern all aspects of our system model in the remaining discussion. A curious-and-faulty server counts as one curious server and

one faulty server. The first assumption implies that there can be many independent groups of colluding servers in the system so long as the maximum number of servers in each group does not exceed t. As a special case, each *individual* server in the system can be curious. The second assumption defines the fault tolerance capacity of the system. When f is zero, the system provides a PIR service.

A faulty server may exhibit coordinated arbitrary and/or malicious behaviours with other faulty servers under the control of an active attacker. For example, an attack may control the faulty servers to: crash, stop responding, modify the content of controlled databases in a coordinated way, and/or deliver purposefully manipulated answers. Apart from conducting the above attacks, the attacker may control curious-and-faulty servers to exchange messages with other curious servers, attempting to find out the user's intention. There is no restriction on the computational capability of the attacker. Hence, the user's privacy is guaranteed in an information theoretic sense. In short, the attacker can choose whatever strategies they like to attack the servers and perform attacks in the way they decide so long as the assumptions of the system are satisfied.

Due to the synchronous system model considered, timeouts can be used to detect the benign behaviours (e.g. crashed servers, send/receive omission) of faulty servers. In particular, if a client does not receive a response to a query message within 2δ time units after sending it, the client can conclude that the server is crashed.

Having defined the system model the ATIR aims to deploy, we are now ready to define ATIR schemes.

3.2.4 Requirements of ATIR

Based on the system model, this section presents two definitions of an ATIR scheme. Before going any further, we first describe five requirements of an ATIR scheme as follows:

1) ***Efficiency***: ATIR schemes should have a non-trivial communication complexity.

2) ***Privacy***: ATIR schemes should maintain the privacy of the user in an information theoretic sense.

3) ***Availability***: ATIR schemes should ensure the availability of a correct result.

4) ***Safety***: ATIR schemes should ensure the correctness of an output if there is an output.

5) ***Liveness***: ATIR schemes eventually terminate.

The efficiency requirement is a *fundamental* requirement for constructing ATIR schemes. Without satisfying the efficiency requirement, a trivial but inefficient solution with the use of k servers can be derived as follows, where $k \geq 2t + 1$. The client can download all k databases and perform all operations locally without using any other services. In this case, the privacy of the user is ensured because every client operation is performed locally. Also, the client can identify the correct result, for example, using majority voting techniques even in the presence of up to t maliciously faulty servers. Due to the condition $k \geq 2t + 1$, the client is guaranteed to be able to get the correct result. Hence, the availability and safety requirements are met. Because of the synchronous setting, the system eventually terminates.

The number of communicating bits this trivial solution is $k \times n \times \log_2 X$ and the communication complexity is $O(n)$. The communication complexity of the solution grows linearly as the number of records increases. Therefore, to have a better solution, ATIR schemes should have a lower than $O(n)$ communication complexity, i.e., sub-linear communication complexity. In other words, using ATIR schemes should be better than downloading all databases and executing the queries locally.

The privacy and availability requirements should be satisfied in normal and faulty circumstances. In the system, the user's private input i is the information that the scheme aims to protect. The privacy requirement ensures that the system keeps i secret from all replicas whereas the availability requirement ensure that at least a correct result can be reconstructed.

The safety requirement is important for fault tolerant schemes [Sch93]. In some circumstances, no output is better than a wrong result because wrong results may lead to disastrous and unpredictable outcomes. The liveness requirement is essential for any distributed schemes [Sch93]. Without satisfying this requirement, the system may not stop and the user may wait forever for a result to be returned.

3.2.5 Definitions of ATIR

This section presents two different definitions of ATIR schemes. The first definition is probabilistic which ensures the safety requirement within a predefined probability. The second definition ensures the safety requirement with a probability of one; hence it is deterministic.

Definition 3.1 (probabilistic ATIR - pATIR) A (k, t, f, ε) pATIR scheme is an algorithm tuple $(Q_1, \ldots, Q_k, A_1, \ldots, A_k, \Re, \Im)$, where k, t, f, ε follow the definitions given in Section 3.2.1. The algorithms are formally detailed as follows:

k Query Algorithms $Q_1, \ldots, Q_k$:

$$\{1, 2, \ldots, n\} \times Z_p^{Lr} \mapsto Z_p^{Lq}$$

k Answer Algorithms $A_1, \ldots, A_k$:

$$\{0, 1, \ldots, X\}^n \times Z_p^{Lq} \mapsto Z_p^{La}$$

A Reconstruction Algorithm $\Re$:

$$Z_p^{Lr} \times (Z_p^{La})^{t+1} \mapsto Z_p$$

A Result Verification Algorithm $\Im$:

$$(Z_p)^k \mapsto \{0, 1, \ldots, X\}$$

The scheme should have the following properties:

a) **Availability**: For $\forall x \in \{0, 1, \ldots, X\}$, $i \in \{1, 2, \ldots, n\}$, and $r \in Z_p^{Lr}$,
$\exists s_1, \ldots, s_{t+1} \in \{1, 2, \ldots, k\}$

$$\Re(r, i, A_{s1}(x, Q_{s1}(r, i)), \ldots, A_{st+1}(x, Q_{st+1}(r, i))) = x_i.$$

b) **Privacy**: For $\forall i, j \in \{1, 2, \ldots, n\}$ and $\forall s_1, \ldots, s_t \in \{1, 2, \ldots, k\}$, and $Q \in Z_p^{Lq}$

$$\boldsymbol{Pr}((Q_{s1}(i, r), \ldots, Q_{st}(i, r)) = Q) = \boldsymbol{Pr}((Q_{s1}(j, r), \ldots, Q_{st}(j, r)) = Q),$$

where the probabilities $\boldsymbol{Pr}$'s are taken over uniformly and randomly chosen $r \in Z_p^{Lr}$.

c) **Safety**: The scheme outputs x_i with a probability no less than $1 - \varepsilon$.

d) **Liveness**: The scheme eventually terminates.

Except the description of safety property, the definition of a deterministic ATIR scheme repeats most parts of that of the probabilistic one. For completeness of presentation, we include the whole definition of a deterministic ATIR scheme here.

Definition 3.2 (deterministic ATIR – dATIR) A (k, t, f) dTIR scheme is an algorithm tuple $(Q_1, \ldots, Q_k, A_1, \ldots, A_k, \Re, \Im)$, where k, t, f follow the definitions given in Section 3.2.1. The algorithms are formally described as follows.

k Query Algorithms $Q_1, \ldots, Q_k$:

$$\{1, 2, \ldots, n\} \times Z_p^{Lr} \mapsto Z_p^{Lq}$$

k Answer Algorithms $A_1, \ldots, A_k$:

$$\{0, 1, \ldots, X\}^n \times Z_p^{Lq} \mapsto Z_p^{La}$$

A Reconstruction Algorithm $\Re$:

$$Z_p^{Lr} \times (Z_p^{La})^{t+1} \mapsto Z_p$$

A Result Verification Algorithm $\Im$:

$$(Z_p)^k \mapsto \{0, 1, \ldots, X, \varnothing\}$$

The scheme should have the following properties:

a) **Availability**: For $\forall\ x \in \{0, 1, \ldots, X\}$, $i \in \{1, 2, \ldots, n\}$, and $r \in Z_p^{Lr}$, $\exists\ s_1, \ldots, s_{t+1} \in \{1, 2, \ldots, k\}$

$$\Re(r, i, A_{s1}(x, Q_{s1}(r, i)), \ldots, A_{st+1}(x, Q_{st+1}(r, i))) = x_i.$$

b) **Privacy**: For $\forall\ i, j \in \{1, 2, \ldots, n\}$ and $\forall\ s_1, \ldots, s_t \in \{1, 2, \ldots, k\}$, and $Q \in Z_p^{Lq}$

$$\boldsymbol{Pr}((Q_{s1}(i, r), \ldots, Q_{st}(i, r)) = Q) = \boldsymbol{Pr}((Q_{s1}(j, r), \ldots, Q_{st}(j, r)) = Q),$$

where the probabilities $\boldsymbol{Pr}$'s are taken over uniformly and randomly chosen $r \in Z_p^{Lr}$.

c) **Safety**: The scheme outputs, if there is an output, the intended result x_i with a probability one.

d) **Liveness**: The scheme eventually terminates.

Similar to traditional definitions of PIR schemes (e.g. the ones presented in [CGKS98] and [BS02]), the above definitions of ATIR schemes use information theoretic privacy property, where a computationally unbounded attacker can gain no information about the user's intention. However, ATIR schemes differ from existing PIR schemes in three major ways. First, the former is based on an active attack model whereas the latter is based on a passive attack model. Second, the former uses a generalised character string database model rather than the standard bit database model. The generalisation is trivial in theory but important in practice. The

generalised model paves the way for implementing ATIR schemes on existing database technologies. Third, ATIR schemes are required to satisfy safety and liveness properties, which are not the requirements in defining PIR schemes.

We now explain the implications of the above properties for *both* definitions of ATIR schemes. The availability property states that there exists at least one set of servers whose answers can be used to reconstruct the correct result. It is important to note that this property only ensures the *existence* of at least one correct result. In the worse case circumstances (i.e., $f = t$), there is only one correct result.

The privacy property means that from any set of t queries, it is impossible to decide which specific record the user is interested in since the joint distribution of random variables $Q_{s1}(i, r)$, ..., $Q_{st}(i, r)$ is independent of i. In other words, from t or less queries, it is theoretically impossible for a server or a group of servers to gain any information about i. The privacy property of both ATIR schemes stems from existing PIR definitions (e.g., the one in [CGKS98]).

The result verification algorithms in pATIR and dATIR guarantee the safety property. For a pATIR scheme, this property guarantees that the scheme can always deliver a result and the result is the intended one with a probability of no less than $1 - \varepsilon$. Whereas for a dATIR scheme, it ensures that the outputted result, if there is one, is x_i with a probability of one.

The liveness property holds provided that $k \geq t + 1 + f$.

Definition 3.3 (Communication Complexity) The communication complexity C of an ATIR scheme is defined as the total number of bits exchanged between the client and all servers. Let C_q be the number of bits sent from the client to a server and C_a be the number of bits sent from a server back to the client. The communication complexity of an ATIR scheme is $k \cdot (C_q + C_a)$.

3.3 Attack-Tolerant Information Retrieval Schemes

In this section, two ATIR schemes are presented, and the presentation is organised as follows: Section 3.3.1 presents three basic algorithms for both schemes. Section 3.3.2 characterises the fault tolerance conditions for these schemes. Section 3.3.3 explains the difficulties of applying conventional error detection mechanisms in ATIR. Section 3.3.4 presents a probabilistic error detection function that is used to detect

incorrect results. Section 3.3.5 describes two result verification algorithms for identifying correct results. Section 3.3.6 presents the formal proofs for both ATIR schemes. Section 3.3.7 applies a generic balancing technique to reduce the communication complexity to $O(n^{1/2})$.

Both ATIR schemes are one-round. That is, a client sends queries to k servers respectively and each server replies with an answer. Based on some of the answers, the client outputs a result. Both ATIR schemes are an extension of the polynomial interpolation based PIR schemes presented by Chor et. al. [CGKS95] in an active attack model.

3.3.1 Basic Algorithms

This section presents three basic algorithms used by both ATIR schemes: a query algorithm, an answer algorithm, and a reconstruction algorithm. These algorithms are based on the polynomial-interpolation PIR schemes presented in [CGKS95].

A Query Algorithm

This section describes a query algorithm which is repeatedly used to create k random and independent queries. The core of the algorithm is the creation and evaluation of polynomial-based query functions. The algorithm consists of four steps. In the first step, the client transforms the user's input into a sequence of bits. In the second step, these bits subsequently are used as the constant terms of the query functions. Note that the other coefficiencies of the polynomials are random elements chosen from the finite field Z_p. Once the query functions are ready, the polynomials are evaluated at a set of distinct points independently chosen from Z_p. This is step three. Along with p, the evaluated results are grouped into k sets of query tuples and these tuples are subsequently sent to the servers, respectively. Figure 3-1 presents the details of the query algorithm.

An Answer Algorithm

Upon receiving a query, a server executes the answer algorithm to calculate an answer based on the answer function. Each answer function calculates the scalar products of two tuples: a query tuple and a database tuple. Since the query tuple is

the query function evaluated at various points, an answer function can be expressed as the following:

$$A(z) = \sum_{j=1}^{n} Q_j(z) \cdot x_j \pmod{p}, \text{ where } z \text{ is an indeterminate over } Z_p.$$

The answer functions are identical for all servers. Figure 3-2 presents an answer algorithm for server S_d.

A Query Algorithm for ATIR

Input: a prime number p; an integer i, where $i \in \{1, \ldots, n\}$; a set randomly and uniformly chosen numbers $r = \{r_{11}, \ldots, r_{1t}, \ldots, r_{n1}, \ldots, r_{nt}\}$, where $r \in Z_p^{nt}$; and k randomly and uniformly selected non-zero distinct numbers $m = \{m_1, m_2, \ldots, m_k\}$ from Z_p.

Output: k random and independent sets with elements in Z_p as queries

1. Map i into a sequence of numbers. For every $i \in \{1, \ldots, n\}$, we define a mapping function h_i: $\{1, \ldots, n\} \mapsto \{0, 1\}$, so that for every $i \in \{1, \ldots, n\}$, $h_i(l) = 1$, if $l = i$; otherwise, $h_i(l) = 0$.
2. Generate n degree-t polynomials as query functions:

$$Q_l(z) \equiv h_i(l) + r_{l1} \cdot z + r_{l2} \cdot z^2 + \ldots + r_{lt} \cdot z^t \pmod{p} \text{ for } l = 1, \ldots, n,$$

where the constant term of the l-th polynomial is $h_i(l)$ and z is an indeterminate over Z_p.

3. Evaluate the polynomials at point $m_1, m_2, \ldots, m_k$ and group the results into k tuples: $< Q_1(m_d), Q_2(m_d), \ldots, Q_n(m_d) >$, where $d = 1, \ldots, k$.
4. Send p and the tuple $< Q_1(m_d), Q_2(m_d), \ldots, Q_n(m_d) >$ as a query to replica S_d for $d = 1, 2, \ldots, k$.

Figure 3-1 A Query Algorithm for ATIR Schemes

An Answer Algorithm for replica S_d

Input: a prime number p, a tuple $Q_d = <Q_{1d}, \ldots, Q_{nd}> \in Z_p^n$ and a tuple $x \in \{0, 1, \ldots, X\}^n$, where Q_d is an tuple extracted from the query for S_d.

Output: an element in Z_p as an answer.

1. Calculate the scalar product of the tuples Q_d and x: $a_d = \sum_{j=1}^{n} Q_{jd} \cdot x_j \pmod p$.
2. Send the element a_d to the client as the answer of S_d.

Figure 3-2 An Answer Algorithm for Replica S_d

A technical lemma about the randomness of answers is proved as follows. In Section 3.3.6, this lemma will be used to prove that the error detection function satisfies certain properties.

Lemma 3.1 Answers in ATIR schemes are randomly and uniformly distributed over Z_p.

Proof: First, we show that answers are randomly and uniformly distributed over Z_p^{La}. Using the query functions, we expand the answer function as follows:

$$A(z) = \sum_{j=1}^{n} Q_j(z) \cdot x_j \pmod p =$$

$$\sum_{j=1}^{n} i(j) \cdot x_j + z \cdot (\sum_{j=1}^{n} r_{j_1} \cdot x_j) + \ldots + z^t \cdot (\sum_{j=1}^{n} r_{j_t} \cdot x_j) \pmod p,$$

when z is fixed, $A(z)$ is a function of the variables $\sum_{j=1}^{n} r_{j_1} \cdot x_j$, ..., $\sum_{j=1}^{n} r_{j_t} \cdot x_j$.

Since all these variables are composed by random variables uniformly chosen from Z_p, $A(z)$ is therefore a function randomly and uniformly distributed over Z_p. As a result, answers are randomly and uniformly distributed over Z_p. *Q.E.D.*

A Reconstruction Algorithm

From the servers, the client receives at least $t + 1$ answers $a_1, \ldots, a_{t+1}$. Based on any $t + 1$ out of the k answers from the servers, the client can perform a reconstruction

algorithm using the polynomial interpolation technique [CLRS01, pp. 826]. Figure 3-3 presents the reconstruction algorithm.

Without loss of generality, let $a_1, \ldots, a_{t+1}$ be a set of $t + 1$ answers that the client chooses to reconstruct a result from. Correspondingly, the client chooses the $t + 1$ distinct points $m_1, \ldots, m_{t+1}$ that have been used to evaluate the query functions for the servers $S_1, \ldots, S_{t+1}$. Then, we have the following set of point-value pairs: $(m_1, a_1), \ldots, (m_{t+1}, a_{t+1})$.

Based on the polynomial interpolation technique, the reconstruction function $\Re$ can be constructed as follows:

$$\Re(z) = \sum_{j=1}^{t+1} a_j \cdot \prod_{\substack{p=1 \\ p \neq j}}^{t+1} \frac{z - m_p}{m_j - m_p} (\mathrm{mod}\, p).$$

The reconstructed value is the evaluation of the reconstruction function at point zero, i.e., $\Re(0)$.

A Reconstruction Algorithm

Input: a set of $t + 1$ non-zero independent and distinct elements $m_1, \ldots, m_{t+1} \in Z_p$ and a separated set of $t + 1$ elements $a_1, \ldots, a_{t+1} \in Z_p$.

Output: an element in Z_p as a result.

1. Evaluate the reconstruction function at zero point:

$$\Re(0) = \sum_{j=1}^{t+1} a_j \cdot \prod_{\substack{p=1 \\ p \neq j}}^{t+1} \frac{0 - m_p}{m_j - m_p} (\mathrm{mod}\, p).$$

2. Output $\Re(0)$ as a result.

Figure 3-3 A Reconstruction Algorithm

3.3.2 Characterisations of Fault Tolerance

This section explains why the existing PIR schemes cannot cope with failures and presents the characterisations of fault tolerance properties of ATIR schemes.

The correctness condition for the polynomial interpolation PIR schemes [CGKS95] is $k \geq t + 1$, where $t \geq 1$. It is both necessary and sufficient for passive server models because an attacker does not change the data and the answers returned.

This condition, however, implies only the existence of a correct result under a passive attack model.

For crashed servers, it follows immediately that

$$k \geq (t+1) + f, \text{ where } t \geq 1 \text{ and } f \geq 0.$$

In the above condition, the parameter t is independent of f. This condition guarantees both the existence of a correct result and tolerance up to f crashed servers. It is also necessary and sufficient when up to f servers do not response (or up to f answers from the servers are lost or intercepted by an attacker during communication). However, based on $t + 1$ or more answers returned, a client could still reconstruct an incorrect result if the faulty servers are malicious. In order to be able to reconstruct a correct result we consider two mutually exclusive hypothesises respectively.

Assumption 1: All the incorrect results derived from the wrong answers returned disagree mutually.

This hypothesis is based on the fact that the wrong answers do not have full control over a reconstructed result provided that $f \leq t$. That is because $t + 1$ answers are required for a reconstruction, when $f \leq t$, at least one correct answer will be used in the reconstruction. In the worse case scenario, t out of $t + 1$ answers are wrong and the remaining one is correct. Hence, it is not possible that a result is reconstructed from the wrong answers only. In other words, any reconstructed result is derived from at least a correct answer from a fault-free server. Hence, the probability that two incorrect results are identical is significantly low. Under Assumption 1, it follows under the malicious server condition that if two reconstructed results are identical, they must be correct, provided that

$$k \geq (t+2) + f \text{ where } f \leq t.$$

Assumption 2: Incorrect results derived from the wrong answers returned may be identical.

This hypothesis states the most general situation and in the worst case all the incorrect results may be identical. In order to tolerate up to f malicious faults, the number of correct results must be always greater than that of potentially incorrect results. We have thus the following theorem.

Theorem 3.1 Under Assumption 2 and the malicious server condition, an ATIR scheme is f-fault tolerant if k, t and f satisfy that

$$2C_{k-f}^{t+1} > C_k^{t+1}.$$

Proof: Note that C_{k-f}^{t+1} is the number of correct results and $C_k^{t+1} - C_{k-f}^{t+1}$ is the number of incorrect results. The number of correct results must be always greater than that of incorrect results, i.e. $C_{k-f}^{t+1} > C_k^{t+1} - C_{k-f}^{t+1}$. Hence, this proves the theorem. *Q.E.D.*

In the worse case, a large number of replicas (approximately in the order of $5f$) are required. This is too costly in practice. However, we notice that the condition $k \geq (t+1)+f$, where $t \geq f$, holds for the malicious server condition and Assumption 2 if a perfect error detection algorithm exists and can be used to identify any incorrect result. The next section explores the construction of such algorithms.

3.3.3 Why is Error Detection Difficult in ATIR?

In fault tolerance, the erroneous state of a system can be identified through acceptance test techniques and/or comparison techniques [HW92]. The goal of error detection is to prevent errors manifesting themselves as system failures. In an acceptance test, a program is executed and its output is subject to a test. If the test is successful, the program continues as normal. Otherwise, an error is signalled. A failed acceptance test is an indication of the presence of a fault. However, the use of acceptance tests alone cannot identify what goes wrong. Combining with other techniques, such as fault masking techniques, correct outputs can be identified and the detected errors can be eliminated.

Traditionally, comparison techniques are mainly used to detect errors caused by software design faults. Errors are detected through comparing the results produced by two or more versions of a piece of software developed by independent organised programming teams implementing a common system specification.

In security, an error often manifests itself as corrupted data. Digital signature techniques, such as the RSA signature [RSA78], can be used to validate whether data has been corrupted during transit. In data communication, garbled messages are detected through error detection codes which typically include checksums and Cyclic Redundancy Checks (CRC) techniques [Bla83].

All these techniques have their own limitations and not all these techniques can be directly applied to the ATIR setting. Acceptance test techniques are general in the sense that they can be applied even if there is only one single component in the system. However, deriving generic acceptance tests is often difficult because the design costs and the effectiveness of acceptance tests are application dependent [Kim95].

Comparison techniques are application independent. For example, voting is a typical comparison technique that can be applied to virtually any applications. But voting is often perceived to be costly due to multiple copies/versions of executions required. Furthermore, the effectiveness of voting is subject to the elimination of common mode failures [AK84]. In an ATIR setting, answers cannot be directly voted on because they are bound to be different even for retrieving the same information.

Digital signatures alone cannot defend against malicious attackers. Besides, the validity of digital signatures relies on the availability of a trusted third party (e.g. certification authorities) before any interactions begin. If messages are corrupted before transmission, digital signatures cannot detect such errors. Finally, error detection codes are designed for identifying errors (e.g., corruption or loss of data) caused during message transition. The challenge of using coding techniques is to overcome the large number of redundancy required.

Designing an effective error detection method has been a great challenge in constructing ATIR schemes. On the one hand, identifying what is correct is hard, given that databases can store arbitrary information. On the other hand, although voting techniques and error detection codes can be employed by ATIR for the purpose of error detection, these techniques are often costly due to the large number of replicas required.

However, error detection helps to identify erroneous statuses but cannot guarantee the delivery of correct results. In order to tolerate malicious attacks, redundancy has to be introduced into the design of an ATIR scheme. Based on existing PIR schemes, our strategy for ATIR is through a combined use of acceptance tests with redundant servers. In our ATIR schemes, acceptance tests identify corrupted results while redundant servers guarantee the existence of correct results. Compared with the techniques described above, this strategy reduces the number of replicas required for error detection and result verification.

3.3.4 Error Detection

It is necessary to emphasise that the condition $k \geq t + f + 1$, where $t \geq f$, implies only the existence of a correct result. In order to identify the correct result as the system output, we have to design a perfect error detection function for ATIR schemes. Alternatively, as Assumption 2 suggests, more replicas can be used to ensure that the client can find a correct result. Correspondingly, more message exchanges are needed and the communication complexity of an ATIR scheme increases. But these solutions are costly.

The polynomial interpolation-based PIR schemes and our ATIR schemes essentially are Shamir's secret sharing scheme [Sha79]. Secret sharing exploits the properties of polynomials (i.e. perfect secrecy and interpolation uniqueness) for providing privacy protection and fault-tolerant operations [GB99]. Our ATIR schemes are based on the same principle. It is therefore possible to apply and extend the existing results of secret sharing, verifiable secret sharing (e.g. [Fel87, CGMA85]), to both PIR and ATIR schemes. In the following, a probabilistic error detection function is developed for ATIR based on the polynomial properties. We now explain the principle of error detection in ATIR.

Principles

The main idea of the probabilistic error detection function is to limit the valid range of reconstructed results. Because a character x_j, where $j \in \{1, 2, \ldots, n\}$, is viewed as an integer taken from a pre-known set $\{0, 1, \ldots, X\}$, for every x_j, there are exactly X candidates of *valid* results. And because all computations are performed over the finite field Z_p, there will be p possible reconstructed results for x_j over the set $\{0, 1, \ldots, p - 1\}$. It follows immediately that if a reconstructed result is within $\{0, 1, \ldots, X\}$, it is valid. Otherwise, it is invalid.

It is desirable that the client has such an error detection capability. That is, when a reconstruction algorithm uses one or more corrupted answers to reconstruct a result, the result becomes invalid. This property of the error detection function is further described in details as follows.

In the following, let us use the worse case circumstance for our analysis. Suppose $(m_1, a'_1), \ldots, (m_t, a'_t), (m_{t+1}, a_{t+1})$ be $t + 1$ point-values pairs that a client uses for

the reconstruction algorithm. For simplicity of presentation, we assume the first t of out the $t + 1$ pairs are corrupted. But in the realistic circumstances, the client does not know which ones are corrupted in advance. In terms of the number of faulty servers, this is the worse case situation because the number of faulty servers is maximum (i.e. $f = t$). It is desirable for the scheme to have the property that there is only a small probability $\varepsilon > 0$ that the output of the reconstruction algorithm is valid but corrupted.

Recall that $\{0, 1, \ldots, X\} \subseteq Z_p$. We can increase the size of Z_p such that most of incorrect results appear in the set $Z_p - \{0, 1, \ldots, X\}$. In other words, the probability of undetected errors can be confined within a pre-defined bound, i.e. undetected error rate ε, for $\varepsilon > 0$. An error is defined as the acceptance of a valid but corrupted result. The error detection function only identifies correct results with a certain probability. Therefore, it is possible that a valid result in $\{0, 1, \ldots, X\}$ is in fact a corrupted result (i.e. not the intended result). However, the user can adjust the probability which is therefore can be arbitrarily close to zero.

The error detection function is simple. If an input element $a \in Z_p$ is also an element in the set $\{0, 1, \ldots, X\}$, the function outputs one (i.e. it is valid); otherwise, it outputs zero (i.e. it is invalid). This function detects corrupted results with a probability. By itself, it does not guarantee perfect error detection, namely the correctness of a reconstructed result is not always guaranteed. Together with the result verification algorithms (which we shall present in the next section), a perfect error detection capability can be obtained.

The possible causes leading for a corrupted result are endless. Here is a list of possibilities: a query or an answer is tampered with in transit or a server does not execute the Answer algorithm properly, one or more answers are corrupted by a malicious attacker. The list is incomplete. In fact it is impossible to enumerate all the possibilities. However, despite the actual causes of corrupted results, the *outcome* is the same: a result is modified so that it is different from the original one. Hence, for simplicity of presentation, we assume that all corrupted results are caused by using corrupted answers in the following proofs.

Determining Z_p

We now can describe how to determine p for both ATIR schemes based on the error detection requirement.

Lemma 3.2 For a (k, t, f) ATIR scheme, suppose p is the smallest prime number and $\varepsilon > 0$ such that the following inequality is satisfied:

$$p > max[X/\varepsilon, k].$$

There is only a small probability $\varepsilon > 0$ that the error detection function accepts a valid but corrupted result.

Proof: Suppose a client receives t corrupted answers $a'_1, \ldots, a'_t$ from replicas $S_1, \ldots, S_t$ and receives one correct a_{t+1} answer from replica S_{t+1}. That is, the client uses $(m_1, a'_1), \ldots, (m_t, a'_t), (m_{t+1}, a_{t+1})$ point-value pairs as inputs for the reconstruction algorithm:

$$\Re(0) = a'_1 \cdot \prod_{p=2}^{t+1} \frac{m_p}{m_p - m_1} + \ldots + a'_t \cdot \prod_{\substack{p=1 \\ p \neq t}}^{t+1} \frac{m_p}{m_p - m_t} + a_{t+1} \cdot \prod_{p=1}^{t} \frac{m_p}{m_p - m_{t+1}}.$$

From Lemma 3.1, we know that answers are randomly and uniformly distributed over Z_p. Despite that $a'_1, \ldots, a'_t$ may be chosen purposefully by an attacker, $\Re(0)$ is a function of a random variable a_{t+1} uniformly distributed over Z_p. Since there are $X + 1$ valid results and only one of them is x_i, there X out of p possibilities that the client reconstructs a valid but corrupted result. By the meaning of ε, the probability of accepting a valid but corrupted results X/p should be at most ε, i.e. $X/p < \varepsilon$. Therefore, we have $p > X/\varepsilon$.

On the other hand, because ATIR schemes also require at least k non-zero points to evaluate the query functions, p should also be greater than k. Hence, this proves the lemma. *Q. E. D.*

It should be made certain that p is large enough to reveal incorrect results with an arbitrarily high probability $(1 - \varepsilon)$, e.g. 99.99%.

Error Detection and Secret Sharing

The principle of the error detection function is the same as the Tompa and Woll's modification [TW88] on the Shamir's secret sharing scheme. Both methods aim to "force" a reconstruction algorithm to reconstruct invalid outputs if the algorithm uses any corrupted inputs. The input in ATIR schemes is called an answer whereas the input in Secret Sharing schemes is called a share. There are differences between both schemes. In ATIR, a client keeps evaluation points locally. Therefore, corrupted

servers can at best guess this information randomly. But in Shamir's scheme, an evaluation point is a part of a share. Hence, corrupted participants have the full information about evaluation points of corrupted shares. Intuitively, an attacker can obtain more information in secret sharing schemes than in ATIR schemes.

Block Error Detection

So far, the error detection function is performed on an individual character basis. In practice a result usually contains a block (i.e., string) of characters rather than just one character. Therefore, the single character error detection function (hereafter, refer to as a single detection) needs to be extended to a block error detection function (hereafter, refer to as a whole detection). In order for a block to be valid, all its characters are required to be valid. Any single invalid character invalids the entire block and a failed reconstruction is resulted by the first unsuccessful single verification. Therefore, the undetected error rate of a whole detection is calculated by multiplying those of single verifications.

Let E denote the undetected error rate of the verification of an entire result and L_a be the number of invalid characters in the result. We have

$$E = (e)^{La}.$$

Instead of specifying ε for each individual character, the user now specifies E as the undetected error rate for the entire result. It is then up to the client program to automatically calculate ε. Since the relationship between ε and E is exponential, a slight modification of ε can significantly change E. Even a user requires a highly effective error detection function by asking E to be 0.05. Assuming L_a is 100, setting ε to be 0.95 is sufficient to satisfy E. Recall that the computational range p is inversely proportional to ε. A slight increase of p can largely improve the fault detection capability of the scheme at a whole.

Implications of the Error Detection Function

This error detection function does not rely on any unproven cryptographic premise, such as, intractability of factorisation of big primes and on the availability tamper-proof hardware (e.g. secure co-processors).

3.3.5 Two Result Verification Algorithms

The section presents two result verification algorithms: one for the pATIR scheme and the other for the dATIR scheme. Both algorithms are used to identify correct results and eliminate incorrect ones. They are both built on the error detection function and the result reconstruction algorithm presented in the previous sections. The result verification algorithms are executed by the client after executing the query algorithm.

Figure 3-4 presents the probabilistic result verification algorithm. After sending queries, the client waits for the availability of at least $t + 1$ answers returned from the servers (line 3). This algorithm stops whenever a valid result becomes available. In the normal circumstance, the algorithm only performs the reconstruction algorithm (line 5) once. In the worse case (i.e. the occurrence of t corrupted answers), the algorithm executes the while loop $\binom{k}{t+1}$ times. This is the situation where corrupted answers are used in every result reconstruction except the last one. The while loop stops at another two situations as well. First, when the algorithm attempts all $\binom{k}{t+1}$ reconstructions but no valid result is found, that implies there are more than t corrupted answers available (line 10). Second, there are more than t crashed servers. In this case, the timeout limit (line 9) is exceeded. Neither of these situations may occur in our system because of the assumptions set out in the system model of the scheme (i.e. the upper bound of f is t).

Figure 3-5 presents the deterministic result verification algorithm. When the algorithm is executed, there are two possibilities: i) all k answers become available within the known time bound; and ii) only a part of the k answers become available within the known time bound. Possibility i) occurs when every server is available. Possibility ii) occurs when there are crashed servers. $\binom{k}{t+1}$ results will be in the former situation whereas less than $\binom{k}{t+1}$ in the latter. No matter which situation actually occurs, the loop (line 11-16) iterates through all the reconstructed results in the set E to check whether they are identical. The identical result is deemed to be the final result and outputted to the user. In the most conditions (including the worse),

the algorithm executes the while loop $\binom{k}{t+1}$ times (line 5). But in the case that there are t crashed servers, the while loop is only executed once. The timeout checking (line 10) in this algorithm has a much important role than that of the probabilistic algorithm in the presence of crashed servers. It is used to prevent the algorithm halts for ever in the presence of even *just one* crashed servers.

Clearly, the probabilistic result verification algorithm is efficient than its deterministic counterpart in normal circumstances. Both algorithms, however, require an equal number of reconstructions in the worse case situation.

A Probabilistic Result Verification Algorithm

1. **set** counter = 0
2. **set-timeout-to** local_clock + 2δ
3. **wait-for** at least $t + 1$ answers returned from the servers
4. **while** (counter $\leq \binom{k}{t+1}$) **do**
5. **select** and **use** a new group of $t + 1$ available answers to **reconstruct** a result *res*
6. **set** counter = counter + 1 % increase the number of reconstructions %
7. **if** (*res* $\in \{0, 1, \ldots, X\}$) **then exit loop**
8. **check** the availability of new answers
9. **on-timeout exit loop**
10. **if** (counter $> \binom{k}{t+1}$ or timeout) **then output** "no result" % more than t corrupted answers or more than t crashed servers %
11. **else output** *res*
12. **stop**.

Figure 3-4 A Probabilistic Result Verification Algorithm

A Deterministic Result Verification Algorithm

Variables: success: Boolean

1. **set** $E \leftarrow \varnothing$, success $\leftarrow$ true, counter = 0
2. **set-timeout-to** local_clock + 2δ
3. **wait-for** at least $t + 1$ answers available
4. **while** (true) **do**
5. **select** and **use** a new group of $t + 1$ available answers to **reconstruct** a result *res*
6. **set** counter = counter + 1 % increase the number of reconstructions %
7. **if** ($res \in \{0, 1, \ldots, X\}$) **then** $E \leftarrow \{res\} \cup E$
8. **if** (counter == $\binom{k}{t+1}$) **then exit loop**
9. **check** the availability of new answers
10. **on-timeout exit loop**
11. **for** $i = 1$ to $|E|$ **do**
12. **for** $j = 1$ to $|E|$ **do**
13. **if** (e_i != e_j) **then**
14. **set** success $\leftarrow$ false
15. **exit loop**
16. **if** (!success) **then exit loop**
17. **if** (success) **then output** *res*
18. **else output** "no result"
19. **stop.**

Figure 3-5 A Deterministic Result Verification Algorithm

3.3.6 Proofs for the ATIR Schemes

After presenting all the algorithms, we are now ready to prove the properties of the ATIR schemes.

We include the Lagrange Interpolation Theorem here because it plays an essential role in proving the properties of both ATIR schemes. For a proof, readers are referred to [LN97, pp. 28].

Theorem (Lagrange Interpolation Formula)

For $n \geq 0$ and any field F, let $a_0, \ldots, a_n$ be $n + 1$ distinct elements of F, and let $b_0, \ldots, b_n$ be $n + 1$ arbitrary elements of F. Then there exists exactly one polynomial $f(x) \in F[x]$ of degree $\leq n$ such that $f(a_i) = b_i$, for $i = 0, \ldots, n$. This polynomial is given by

$$f(x) = \sum_{i=0}^{n} b_i \prod_{\substack{k=0 \\ k \neq i}}^{n} (a_i - a_k)^{-1}(x - a_k).$$

In the following proofs, we use a special case of the Lagrange Interpolation Formula by considering the field F be a finite prime field Z_p.

Before discussing the properties, we formally prove some technical lemmas about the answer function and the reconstruction function. Lemma 3.3 shows that the answer function is effectively identical to the reconstruction function. Hence, they can be used interchangeably in proving the theoretical results. The Lemma 3.4 shows that the constant term of the reconstruction function is the intended result when the answers used for the reconstruction are correct.

Lemma 3.3 Without loss of generality, let $(m_1, a_1), \ldots, (m_{t+1}, a_{t+1})$ be a set of correct point-value pairs, where $m_1, \ldots, m_{t+1}$ are $t + 1$ distinct elements $\in Z_p$ and $a_1, \ldots, a_{t+1}$ are correct answers. The answer function is identical to the reconstruction function. And the degree of the answer function $A(z)$ (or $\Re(z)$) is at most t.

Proof: Since the answer function is identical for all servers, from its construction, we can view the values $a_1, \ldots, a_{t+1}$ as the results of evaluating the function at points $m_1, \ldots, m_{t+1}$ respectively. On the other hand, $(m_1, a_1), \ldots, (m_{t+1}, a_{t+1})$ are used as the inputs to the reconstruction function. Since $m_1, \ldots, m_{t+1}$ are $t + 1$ distinct elements $\in Z_p$ and $a_1, \ldots, a_{t+1} \in Z_p$, due to the Lagrange Interpolation Theorem, we

know that these $t + 1$ point-value pairs $(m_1, a_1), \ldots, (m_{t+1}, a_{t+1})$ uniquely determines a polynomial of degree at most t. Hence, the answer function is identical to the reconstruction function and its degree is at most t. Hence, this completes the proof. *Q. E. D.*

Lemma 3.4 When a reconstruction function $\Re$ is created by $t + 1$ correct point-value pairs, the constant term of $\Re(z)$, i.e., $\Re(0)$, is x_i.

Proof: From Lemma 3.3, we know that $A(0) = \Re(0)$. Therefore, it is sufficient to show that $A(0)$ is x_i. From the construction of the answer function, we know that

$$A(0) = \sum_{j=1}^{n} g_j(0) \cdot x_j \pmod{p} = \sum_{j=1}^{n} i(j) \cdot x_j \pmod{p}.$$

From the construction of the mapping function presented in Section 3.3.1, we know that when $i \neq j$, $i(j) = 0$ and $i = j$, $i(j) = 1$. Therefore, we have $A(0) = x_i$. This proves the Lemma. *Q. E. D.*

Proofs for the pATIR Scheme

We are now ready to prove the properties of ATIR schemes. We first present the theoretical results about pATIR schemes.

Theorem 3.2 (pATIR Availability) The pATIR scheme satisfies the availability property.

Proof: It is sufficient to show that the scheme guarantees the existence of $t + 1$ correct answers despite attacks. According to the system model described in Section 3.2.1, there are only up to f faulty servers in the system, $k \geq t + 1 + f$ and f is bounded by t. Hence, the client can receive at least $t + 1$ correct answers. Due to the synchronous setting of the system, the answers from these $t + 1$ servers are guaranteed to be received by the client within a known bounded time. By Lemma 3.4, we know that these answers can be used to reconstruct x_i.

Hence, the conclusion. *Q. E. D.*

Theorem 3.3 (pATIR Privacy) The pATIR scheme satisfies the privacy property.

Proof: According to the definition of the privacy property of a pATIR scheme, it is sufficient to show that the information of no more than t queries reveals nothing about i. That is, the joint distribution of t queries is independent of i. Recall that the

query functions are polynomials of degree at most t. According to the Lagrange Interpolation theorem, for a polynomial of degree at most t, t (or fewer) distinct point-value pairs indicate no information about the polynomial. Hence, from t (or fewer) points, no information about its free term can be obtained either. Note that any of these polynomials can be the one that are used for generating the queries. Due to the assumption that there are no more than t curious servers who collude together, an attacker can at most collect the information about the polynomial at t points. Therefore, no information about i can be revealed from no more than t queries. Hence, the conclusion. *Q.E.D.*

Theorem 3.4 (pATIR Safety) The pATIR scheme satisfies the safety property.

Proof: Lemma 3.2 indicates that there is only a small possibility $\varepsilon > 0$ that the error detection function fails to detect a corrupted (but valid) result. Additionally the probabilistic result verification algorithm presented in Figure 3.4 indicates that when the pATIR scheme outputs a result, there are only two possibilities: i) the result is correct; or ii) the result is valid but corrupted with a small probability ε. Hence, the scheme outputs a correct result x_i with a probability no less than $1 - \varepsilon$. Therefore, we have the conclusion. *Q. E. D.*

Theorem 3.5 (pATIR Liveness) The pATIR scheme satisfies the liveness property.

Proof: In order to prove the liveness property, it is sufficient to show that the probabilistic result verification algorithm stops. Theorem 3.2 indicates that the scheme guarantees the existence of at least one correct result. The existence of this result ensures that the algorithm will stop. Hence, the conclusion. *Q.E.D.*

Proofs for the dATIR Scheme

The dATIR scheme employs the same set of basic algorithms as the pATIR scheme but uses different result verification algorithms. Because the result verification is locally performed by the client, the proofs of the availability and privacy properties of the dATIR scheme remain exactly the same as with the pATIR scheme. The proofs of the safety and liveness properties of the dATIR scheme, however, are different.

Theorem 3.6 (dATIR Availability) The dATIR scheme satisfies the availability property.

Proof: Since this proof is exactly the same as the proof given for Theorem 3.2, the details are therefore omitted.

Theorem 3.7 (dATIR Privacy) The dTIR scheme satisfies the privacy property.

Proof: Since this proof is exactly the same as the proof given for Theorem 3.3, the details are therefore omitted.

Theorem 3.8 (dATIR Safety) The dTIR scheme satisfies the safety property.

Proof: Since the error detection function may fail to detect valid but corrupted results. At the end of the dATIR scheme, there is a chance that two or more distinct valid results are available. In this case, the scheme outputs no result to avoid outputting a corrupted result. On the other hand, after all the iterations, if all results are identical and valid, it must be a correct result. Therefore, the outputted result, if there is one, must be the intended one. Hence, the safety property is ensured. *Q.E.D.*

Theorem 3.9 (dATIR Liveness) The dATIR scheme satisfies the liveness property.

Proof: To prove the liveness of the scheme, it suffices to show that the deterministic result verification algorithm stops in the following situations: i) all answers are returned from the servers; and ii) only a part of the k answers are returned. In the first situation, the while loop stops at line 8 and $\binom{k}{t+1}$ results are reconstructed. This situation covers both normal circumstances and the circumstances that there are faulty (but not crashed) servers. All these results are subject to the second loop (line 11 – 16), which stops after $|E|^2$ loops in the worse case. The second situation occurs in the presence of at least one crashed server. This also includes the situation that some answers used for the reconstruction algorithm are corrupted. The algorithm stops when the algorithm timeouts (line 10). Again, all the results are subject to the second loop (line 11 – 16), which stops after $|E|^2$ loops in the worse case scenario. Hence, the conclusion. *Q.E.D.*

Now, we have presented all the theoretical results of both ATIR schemes. However, the communication complexity of both schemes is $O(n)$. In the next section, we reduce the communication complexity to $O(n^{1/2})$ through a generic balancing technique which is first discussed in [CGKS95] and latter widely used in

several other efforts, such as that presented in [CG97], to reduce the communication complexity of their schemes.

3.3.7 Communication Complexity

The communication complexity in both ATIR schemes is unbalanced. The client sends $n \cdot log_2 p$ bits to each server whereas the server replies with one single element with $log_2 p$ bits. A similar problem was first observed by Chor et. al. in [CGKS95] on PIR schemes. Note that the technique they proposed is based on the binary bit database model whereas ATIR schemes use a character string database model. This section shows how to apply a generic balancing technique presented in [CGKS95] to balance the bits exchanged between the client and the server.

The generic balancing technique approaches this problem by partitioning the characters of the database into m blocks $B_1, \ldots, B_m$. Each block contains l characters. Without loss of generality, let use assume $n = m \cdot l$. (the database is padded with dummy values when necessary.) Instead of applying the answer algorithm of an ATIR scheme to the entire database x, the algorithm is now *repeatedly* applied to each block. (In practice, this process can be done concurrently.) The index i is converted into an index i', which is its relative position in the corresponding block. The new position is calculated by $i' = i \pmod{l}$. That is, i' is now an element of the set $\{1, \ldots, l\}$. The query functions are constructed using i' as an input. As a result, a query consists of l elements. This query is repeatedly used by the algorithm on each block of the database. Consequently, instead of containing one element, an answer now consists of m elements, one for each block. The rest of the ATIR scheme remains the same.

After using the balancing technique, the communication complexity of the ATIR schemes is brought down to $O(n^{1/2})$. $k \cdot l \cdot log_2 p$ bits are sent from the client to the servers while $k \cdot m \cdot log_2 p$ bits are sent back from the servers. In total, there are $\log_2 p \cdot k \cdot (m + l)$ bits exchanged between the client and the servers. When $m = l$, the communication complexity is $2 \cdot \log_2 p \cdot k \cdot n^{1/2}$.

3.4 Comparisons of ATIR with Existing PIR Schemes

3.4.1 Comparing with Robust PIR Schemes

The closest work related to ATIR is Beimel and Stahl's *robust Private Information Retrieval* (abbreviated rPIR henceforth) schemes, which were presented in [BS02]. Their schemes consider an active attack model and use Reed-Solomon Codes (RSC) [RS60, MS81] to identify correct results. The principle of RSC is based on majority voting techniques in fault tolerance. ATIR schemes differ from the rPIR schemes in several key aspects. Firstly, the correctness condition for rPIR schemes is $k \geq 3t + 1$ whereas that of ATIR schemes is $k \geq t + 1 + f$ where $t \geq 1$ and $f \leq t$. In order to tolerate up to f maliciously faulty servers where $f \leq t$, ATIR schemes use roughly 2/3 servers that rPIR schemes need. In practice, the reduction of the number of replicated servers is significant because replication is costly. This is because their schemes only rely on majority voting to identify the correct results whereas ATIR schemes use probabilistic error detection to eliminate incorrect results, and rely on the result verification algorithms to identify the correct ones.

In rPIR schemes with a crash failure model, the answers from any k out of l servers are sufficient to reconstruct a result and up to t servers are allowed to collude with each other. Such schemes are called t-private k-out-of-l robust PIR schemes, where $l \geq k$, $t \geq 1$ and $k \geq t + 1$, and the communication complexity of these schemes is $O(\frac{k}{t} \cdot n^{t/k} \cdot l \cdot \log l)$. In particular, under the same model, the authors also construct 2-out-of-l robust PIR schemes with $O(n^{1/3} \cdot \log l)$ communication complexity. But no communication is allowed among the servers.

In ATIR schemes with a crash failure model, the answers from any $t + 1$ out of k servers can be used to reconstruct a correct result, provided that the number of colluding servers is bounded by t, there are up to f crashed servers, and $k \geq t + f + 1$. These ATIR schemes have an $O(n^{1/2})$ communication complexity.

In a malicious failure model, two types of Byzantine rPIR schemes are constructed. If no communication is allowed among the servers, a t-Byzantine robust k-out-of-l PIR scheme is presented with an $O(k \cdot n^{1/\lfloor k/3 \rfloor} \cdot l \cdot \log l)$ communication complexity, where $t < \lfloor k/3 \rfloor$, $t \geq 1$, $l \geq k$.

However, the assumption that there is no communication among compromised servers (still in the malicious failure model) is not realistic. Therefore, the authors further present a t-private and t-Byzantine robust k-out-of-l PIR scheme which allows the rPIR schemes to tolerate any collusion of up to t servers and up to t Byzantine faulty servers. The communication complexity of the t-private and t-Byzantine rPIR schemes is $O(\frac{k}{3t} \cdot n^{\frac{1}{\lfloor (k-1)/3t \rfloor}} \cdot l \cdot \log l)$ communication complexity, where $t < k/3$, $t \geq 1$, and $l \geq k$. In particular, when t is one, the communication complexity of these rPIR schemes become $O(\frac{4}{3} \cdot n \cdot l \cdot \log l)$, which is a trivial result in terms of communication complexity. Therefore, in order to obtain non-trivial communication complexity, at least five servers (when $t = 1$) have to be used and the corresponding communication complexity is $O(\frac{5}{3} \cdot n^{\frac{1}{4}} \cdot l \cdot \log l)$ [Bei04]. In order to achieve $O(n^{1/2})$ communication complexity, these schemes require at least seven servers.

Under the malicious failure model, ATIR schemes achieve better communication complexity while requiring fewer servers. In particular, we construct t-private and t-malicious ATIR schemes which allow up to any t servers to collude together and tolerate up to f maliciously faulty servers, provided that $k \geq t + f + 1$, $t \geq 1$, and $f \leq t$. The communication complexity of our ATIR schemes is $O(n^{1/2})$. For example, to tolerate two maliciously faulty servers and having an $O(n^{1/2})$ communication complexity, ATIR schemes require at least five servers whereas rPIR schemes require seven.

3.4.2 Comparing with Hardware-based PIR Schemes

With the use of secure co-processors, hardware-based PIR (hPIR) schemes can reduce both computation and communication costs incurred by other PIR schemes. Since hPIR does not take server faults into account, we consider a special case of ATIR where there is no faulty servers in the system, i.e. $f = 0$. In this case, ATIR is reduced to a normal PIR and ATIR can only offer privacy protection for users.

In hPIR schemes, secure co-processors are installed on a PIR server, and are treated as black boxes within which all PIR operations are performed. Each co-processor is assumed to have established a secure channel with a user, for example, using encryption. All the traffic in and out of these secure co-processors passes

through these secure channels. Due to the use of encryption, the server cannot figure out any information (in a computational sense) of PIR queries and answers. Hence, the user's privacy is protected.

Due to the use of encryption and secure co-processors, the communication complexity of all hPIR schemes is brought down to be constant (independent of the size of a database), i.e. O(1). But the computation cost of some early hardware-based PIR schemes [SS00, SS01] is O(n). Therefore, the subsequent work [AF02, IS03, Aso04] introduces advanced pre-processing techniques (i.e. periodical database shuffling) to reduce online computation time to be independent of database sizes, i.e., O(1). However, albeit a constant, the online computation time in these schemes grows linearly as the number of queries increases. Periodical shuffling and encryption operations are required to prepare databases for these hPIR schemes.

Like the majority of PIR schemes, the computation complexity of ATIR schemes is O(n), i.e. the computation time of ATIR grows linearly as the number of records involved increases. In terms of online computation costs, ATIR is much worse than hPIR. However, ATIR does not require *any* database pre-processing and the database in ATIR is available all the time.

Whereas as shown in [IS03, Aso04], hPIR schemes have exceedingly high pre-processing costs (in the order of hours in the best algorithm known so far [Aso04]). That is due to the algorithmic cost of performing periodical shuffling. Since this is a mandatory process for privacy concerns, it can cause performance and deployment concerns in practice. Currently, even using the best shuffling algorithm, the computation complexity of shuffling is O($n^{1.5}$) [Aso04] which is verified experimentally.

However, ATIR relies on k ($\geq$ 2) replicated servers and restricts the communication among servers to provide privacy protection whereas hPIR only requires one single server to achieve the goal.

The communication complexity of both ATIR schemes is O($n^{1/2}$). However, unlike hPIR, ATIR require neither encryption nor secure hardware to support privacy protection. In summary, we believe that ATIR is appealing because it demands fewer configurations on and makes fewer assumptions about on the execution environment.

Limitations of Hardware-based PIR Schemes

Here, we discuss the limitations of hPIR schemes in details. hPIR schemes impose strong trust assumptions on the deployment environment which significantly restrict the practicability of these schemes.

Pre-processing, such as encryption and shuffling, is needed to prepare a database for hPIR operations. The database of an hPIR scheme is assumed to be *encrypted* to hide its content. That is because without encryption, the server can easily spot the identity of a record being retrieved. *Periodical reshuffling* is needed to randomise the positions of the records in the database. Otherwise, the access history can reveal some information about the intended record. Between two reshuffling operations, no database updates are allowed. As a whole, the encryption and shuffling requirements mean that hPIR schemes have to use dedicated databases which are maintained separately from normal databases. In theory, this is feasible. But in reality, it is doubtful whether a service provider will provide such setting just for the sake of the privacy of users.

A trusted copy of the encryption and shuffling algorithm implementations is needed to be installed on secure co-processors before any hPIR schemes start. The communication channels, between secure co-processors and clients, are needed to be encrypted to prevent privacy violation from the servers and the communication channels.

3.4.3 A Summary of Comparison Results

So far, we have presented ATIR schemes and compared them with relevant PIR schemes. Table 3-1 presents a comparison between ATIR schemes and two other most relevant PIR schemes (i.e. rPIR and hPIR). The downloading solution discussed in section 3.2.4 is abbreviated as the dl-PIR in the table. Columns three, four, and five represent the communication complexity, the worse case reconstruction times, and the computation complexity of these schemes, respectively.

Table 3-1 A Comparison of PIR Schemes

Schemes	*# servers*	*Comm. Comp.*	*Rec. Comp.*	*Comp. Comp.*	*FT*	*Privacy*
dl-PIR	$k \geq 2t+1$ $t \geq 0$	$O(n)$	No	No	$\leq t$ malicious servers	Against all servers
dATIR	$k \geq t+f+1$ $t \geq 1, t \geq f$	$O(n^{1/2})$	$\binom{k}{t+1}$	$O(n)$	$\leq t$ malicious servers	Collusions $\leq t$ servers
pATIR	$k \geq t+f+1$ $t \geq 1, t \geq f$	$O(n^{1/2})$	1	$O(n)$	$\leq t$ malicious servers	Collusions $\leq t$ servers
rPIR	$k \geq 3t+1$ $t \geq 1$	$O(n^{\frac{1}{\lfloor (t-1)/3 \rfloor}})$	$\binom{k}{t+1}$	$O(n)$	$\leq t$ malicious servers	Collusions $\leq t$ servers
hPIR	1	O(1)	No	O(1)	No	Against one server

3.5 Discussions

We have presented the construction and theoretical results of two ATIR schemes and compared them with relevant PIR schemes. In order to fully understand the strength and limitations of these schemes and the implications of our results, we discuss some relevant issues of ATIR in a wider context.

3.5.1 Validity of ATIR System Assumptions

We now examine the three assumptions made in the ATIR system model in turn. The first two assumptions place a bound on the number of curious servers and the number of faulty servers. To realise these assumptions, it is important to apply the design diversity approach [AK84] in various stage of system design and implementation. Otherwise, an attacker can easily exploit a common vulnerability of all servers to compromise the entire system. Consequently, the scheme will be of little use if such vulnerability can be easily found and exploited. For example, diverse operating systems (Linux and Windows) and programming languages (e.g. Java and C) can be used for the implementation. We can also choose from a wide range of readily available commercial database engines (e.g. MySQL and Microsoft SQL Server) for the servers. All these countermeasures may help to reduce the overall vulnerability of the system by incorporating diversity into the system implementation. So long as all the implementations conform to a well-defined set of protocol *interfaces* (which we

shall describe in the next chapter), the problem of communicating among diverse implementations can be resolved.

The third assumption is about the trustworthiness of the client throughout the lifetime of the system. For example, this assumption can be realised by requiring the user to choose a personal computer (i.e. a client) which the user has full control of and to ensure the authenticity of the software installed on this computer.

3.5.2 Comparison with Existing Secure and Fault Tolerant Schemes

We now compare ATIR with three latest developed secure and fault tolerant algorithms/systems. These systems are BFT [Cas01, CL99], COCA [ZSR02], and SINTRA [CP02] and they share many similarities with ATIR. For example, all these systems use replicated servers, are based on the active attack model and provide correct services so long as no more than a threshold number of servers are corrupted.

But they also differ from ATIR in several important aspects. All these systems rely on an external trusted party (e.g. a system operator) to setup the systems. In particular, distributing security keys (e.g. authentication keys, encryption keys, signing keys) is mandatory to be performed by a trusted operator. However, the purpose of employing such a trusted party is different in these systems. BFT is a generic fault tolerant algorithm. The use of security keys in BFT is for enabling authentication among participants and providing secure communication within the system (thus thwarting eavesdroppers, for example). The current ATIR system model assumes authenticated but *not* secure communication links. When authentication is needed in ATIR, authentication mechanisms can be added into ATIR as BFT does. However, messages transmitted over the communication links are not required to be encrypted in ATIR.

In COCA and SINTRA, the role of the trusted party is more essential than that in BFT. COCA and SINTRA are application specific systems: COCA aims to provide an online certification authority whereas the goal of SINTRA is to enable secure DNS. Apart from demanding authenticated and secure communication among participants, the services in COCA and SINTRA require the protection of the service signing key, such as the signing key of a certification authority. Both systems use some variants of secret sharing (discussed in Section 2.5.2) to split a signing key and distribute the shares of the secret. Proactive security (discussed in Section 2.5.4) with

the use of secure co-processors is also employed to enable regular refreshment of the key. Therefore, in both systems, the trusted operator is also responsible to distribute the initial shares of the key.

In contrast, no trusted operators are required to initialise an ATIR system. No secure co-processors are needed during the operation stage of an ATIR system either.

Finally, none of the three systems provide fault-tolerant privacy protection as ATIR does. When a server is compromised by an attacker, users' privacy is violated in all three systems. On the contrary, ATIR provides privacy protection even in the presence of active attacks.

ATIR provides a new way of detecting and tolerating malicious attacks without relying on trusted third parties during the setup and operation stages of an ATIR system. However, the semantic of ATIR is much weaker than all three systems. ATIR only provides read-only operations whereas they offer both read and write/update operations.

3.6 Summary

This chapter presents two ATIR schemes for performing database queries in a synchronous distributed network environment. These schemes protect the privacy of users and ensure the correctness of results even in the presence of malicious attacks.

We tackle the ATIR problem through three closely linked techniques: privacy protection, error detection and attack tolerance. By hiding the intention of retrieval operations, the privacy of users can be protected. Hence, the risk of targeted attacks can be reduced. Through restricting the range of valid results, errors may be detected and corrupted servers may therefore be identified. Finally, attack tolerance is achieved through the introduction of a form of redundancy – replication, a classic and well-known fault tolerance technique for tolerating faults. As a whole, all three techniques complement to each other and together they provide a solution for the ATIR problem.

Together with a thorough description of the formal database model, and a list of assumptions used in the ATIR system model, detailed constructions of ATIR schemes are first presented. This is followed by a presentation of the basic algorithms of ATIR schemes and a detailed characterisation of their fault tolerance

conditions. We then describe the rationale for introducing a probabilistic error detection function that is followed by a detailed description of the principle, calculation, extensions, and implications of the error detection function. Two result verification algorithms are then derived: one for probabilistic ATIR and the other for deterministic ATIR. The properties of both ATIR schemes are proved and the communication complexity these schemes are reduced to $O(n^{1/2})$. ATIR is then compared with relevant PIR efforts to reveal the strengths and limitations of ATIR. Finally, we place ATIR in a wider context by discussing the validity of the assumptions made in the ATIR system model and further examining the relationship between ATIR and other state-of-the-art secure and fault tolerant systems.

Chapter 4 System Architecture and Implementation

In the preceding chapters we have presented the theory to model the ATIR problem and to construct ATIR schemes. This chapter describes a system architecture to realise the ATIR schemes based on realistic databases. We also discuss the design decisions and implementation issues that arise during system implementation.

PIR can be viewed as a special case of ATIR in that there is no faulty server in the system. We have also implemented the polynomial-interpolation based PIR schemes presented in [CGKS95]. Since the PIR implementation is very similar to the ATIR implementation, only some distinct details of the former are included.

4.1 System Architecture

At the basic level, an ATIR system can be viewed as a data query service with additional security and fault tolerance supports.

To distinguish ATIR services from the conventional data query services, such as database queries, we illustrate the logical relationship of the layers of the ATIR system in Figure 4-1. However, it is important to note that ATIR only supports read-only operations and an ATIR query has a *much restricted SQL syntax* than general SQL statements. In particular, ATIR users are assumed to know the index information about the records in a database. (Section 4.3.1 discusses some possible solutions of relaxing this assumption.)

At the basic design level, an ATIR system follows the conventional client-server architecture consisting of a client and a server. At the top layer, a user program interacts with database services through the Data Query Protocol (DQP). The DQP protocol is simply an abstracted representation of ATIR query operations that can be invoked by the user. Ideally, apart from specifying their security and fault tolerance requirements, users should not need to worry about the underlying protocol complexity of ATIR systems. To facilitate this need, a set of standardised (i.e.,

following conventional function calls in UNIX/LINUX) interfaces is designed to accommodate a range of such requirements. The system also provides a set of default settings to ease the pain of complicated configuration.

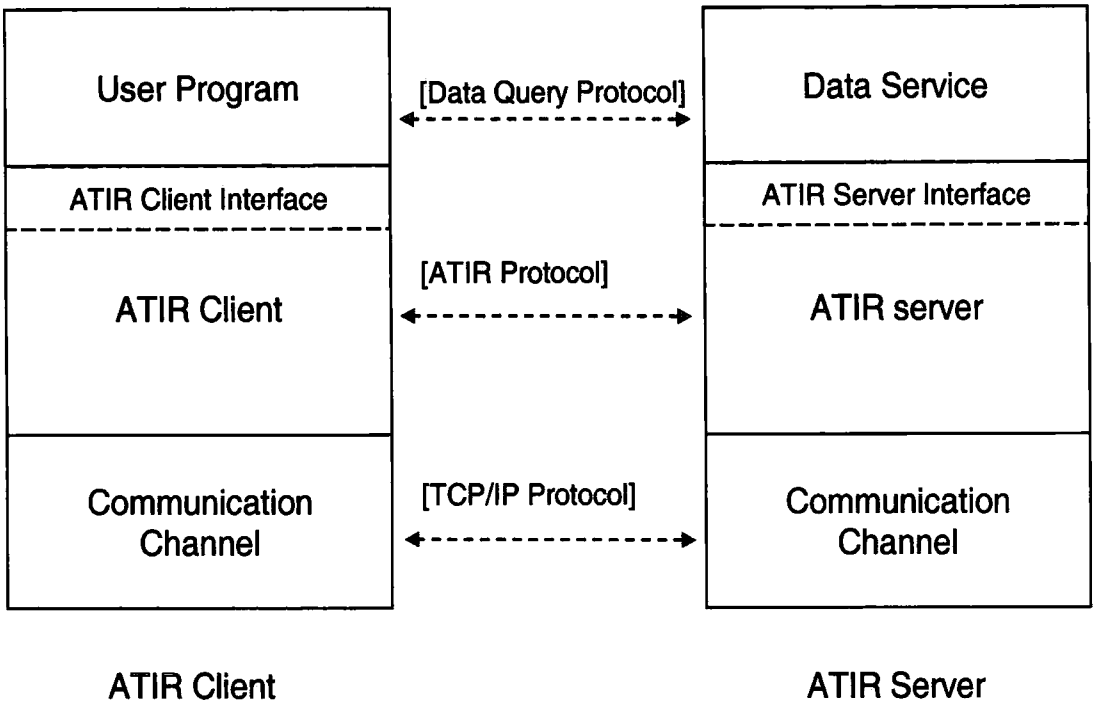


Figure 4-1 The ATIR System and Network Layers

ATIR clients and servers communicate via the ATIR protocol, which specifies the format and types of the data contained in the ATIR queries and answers. We shall give further details of the ATIR protocol in the later section of this chapter. The ATIR protocol sits on top of the TCP/IP communication channels.

In the system, there are two types of Application Program Interfaces (APIs): one for the client and one for the server. The ATIR client interface API defines the type and format of the input that the system takes from the user and the result that the system outputs to the user. The server interface API specifies the type and format of the information that it passes to and obtains from the normal data services, such as databases or data directories. Commonly, a query for these services is required to follow specific protocols and through dedicated data engines or database drivers. The server API interface ensures that the queries created by the ATIR server conform to these protocols and support seamless integration with backend data services.

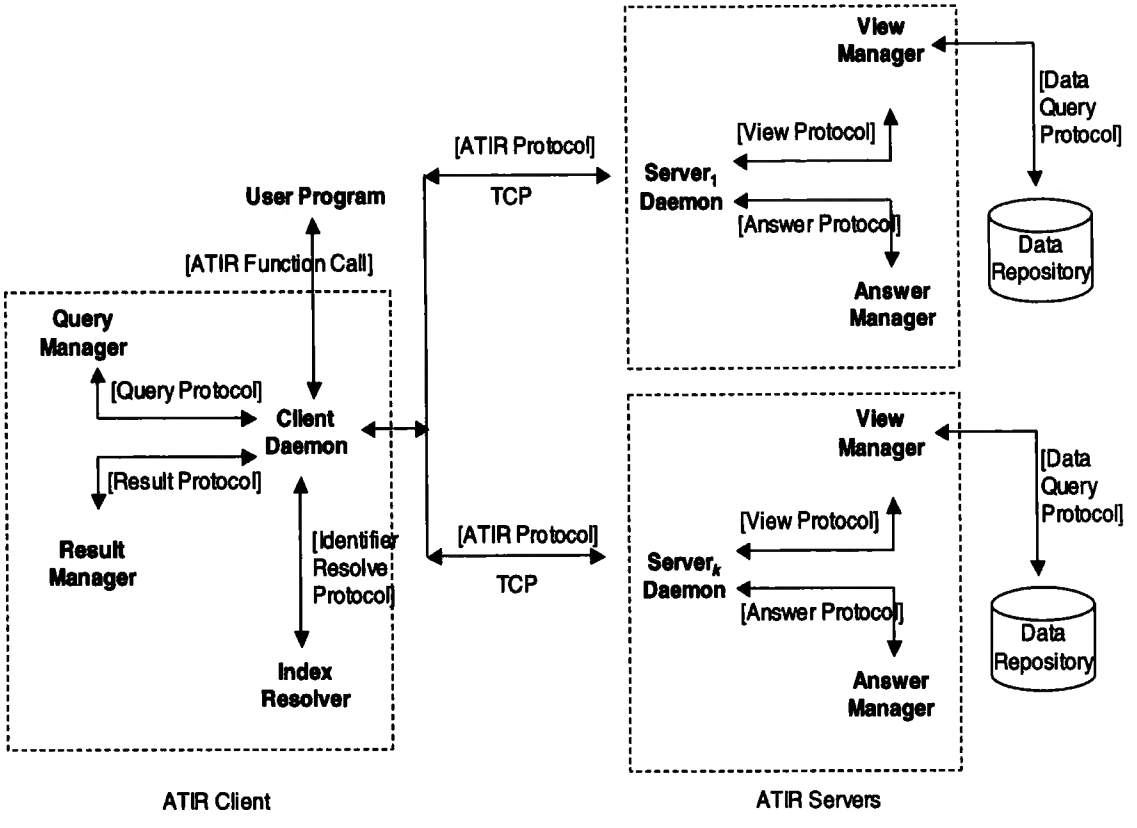


Figure 4-2 An ATIR System Architecture in a Replication Setting

As shown in Figure 4-2, an ATIR system is *conceptually* divided into a number of components by their functionalities. On the client side, we have the following components: client daemon, query manager, result manager, and index resolver. On the other hand, the server side components are server daemon, view manager, and answer manager. Some of the components are integrated with the others to avoid the unnecessary complications in real implementation. For example, the result manager is implemented as a major part of the client program in the current implementation.

ATIR is implemented as a stand-alone library and can be utilised through invoking an ATIR library function. The implementation is currently supported under the Linux environment. The current design, however, can be easily ported to different platforms (e.g., WINDOWS) with little or no modifications. Meanwhile, the modular design may help to ease the effort of porting the ATIR library to Windows operating system as well. The ATIR library provides simple APIs and gives the user (e.g. a program) to choose whether to utilise ATIR service with integrated security and fault tolerance features. We have also implemented a simplified version of ATIR database service in Java to demonstrate the portability of the implementation.

The user invokes the ATIR service through an *ATIR function call* which connects the user with the client daemon *tircd*, which is a central processing program for coordinating the tasks of the client side. *tircd* first decides whether it needs to perform the *Index Resolve Protocol* (*IRP*) to obtain the index of an intended data item from the index resolver. If it does, it resolves the index. Together with the index and other information it already acquires from the user, *tircd* creates a request and forwards it to the query manager *qm* through the *Query Protocol* (*QP*). When the *QP* protocol completes, *tircd* obtains a reply (in the form of a tuple) from *qm* which contains k ATIR queries, one per server. The queries are sent to the ATIR servers appropriately via the *ATIR Protocol* (*ATIRP*) over point-to-point TCP channels.

The ATIR server daemon *tirsd* forwards ATIR queries to a view manager *vm* via the *View Protocol* (*VP*). The *vm* converts the ATIR queries into the appropriate format that is accepted for performing the *Data Access Protocol* (*DAP*) on the data repository on the servers. When the *VP* protocol completes, *tirsd* obtains a tuple that consists of the inputs for the answer computation. In summary, *vm* polls the information from the data repository according to the ATIR queries and forwards such information to the answer manager *am* via the *answer protocol* (*AP*). *am* is responsible for the server side computation of the ATIR scheme. *am* produces an answer which is sent back to *tircd*, again, over the point-to-point TCP channels.

Upon receiving answers, *tircd* forwards them to the result manager *rm* via the *Result Protocol* (*RP*). When the *RP* is completed, the intended result is returned to the user via *tircd*. This completes the ATIR function call.

4.2 Design Issues

4.2.1 The Character-String Database Model

Why Character String Model?

This section describes why and how to extend the existing bit-string database model to character-string database model. In the original PIR database model, each database is modelled as a binary bit string where each bit is an abstraction of a data item on the server. To implement PIR schemes for real applications, we need to extend the model and map the abstracted notation to reality. In the real world, the smallest unit of a data item is commonly represented as a single byte character. Most commercial

database engine, such as MySQL [MySQL04] database server and JDBC [JDBC04] database APIs, provide methods to retrieve data (items) as character strings. In our current implementation, each character is associated with an extended ASCII code and is uniquely associated with a decimal integer in the interval [0, 255].

Having described the rationale for a character-string database model, the next section explains how to associate a character with a unique element of a finite field so that it can be used for the PIR/ATIR computations. First, we show how to associate characters with field elements of finite prime fields. We then describe how to associate characters with field elements of *GF*(256). The former is applicable for our ATIR implementation whereas the latter one is for the PIR implementation only.

Characters, Prime Fields, and ATIR

This section shows the association of characters with field elements in finite prime fields Z_p, where p is a prime number and $p > 255$. In the current implementation, each character can be uniquely associated with a smallest possible element in the chosen prime field. For instance, the character 'A' is associated with the integer 65 in the prime field *GF*(331).

Prime fields, however, are not necessarily for implementing PIR due to the performance concerns. Since there are 256 characters, ideally, the computation should be done with all these characters. Besides, the elements in *GF*(257) consume more memory spaces and communication bandwidth which can be a concern when measuring the message sizes exchanged among the client and the servers. The smallest possible prime field that can accommodate all 256 characters is *GF*(257) which is the set {0, 1, 2, ..., 256}. There is one extra element, i.e., 256, which cannot be associated with any character.

Although we can use *GF*(257) for PIR operations, however, GF(257) is not necessary for the PIR implementation because of the extra overheads that may be introduced by the extra element. It requires 2-byte integers to represent the field elements in *GF*(257) and only 1-byte integer for *GF*(256). That is, from a theoretical point of view, using *GF*(257) doubles the communication bandwidth and storage space.

This suggests that the PIR implementation may only needs a finite field which can just accommodate all 256 elements. Fortunately, we do have such a finite field GF(256) which has 256 elements.

Characters, GF(256), and PIR

Before going any further to explain how to associate the single byte characters with field elements of $GF(256)$, we need some brief background in finite fields. From finite field theory [LN97], we know the following:

$$GF(256) = Z_2[x]/m(x),$$

where $Z_2[x]$ are polynomials over the finite field Z_2 [LN97 pp. 20, Chi79 pp. 125], $m(x)$ is a degree-8 irreducible polynomial [LN97, pp.91] in $Z_2[x]$. The equation means that $GF(256)$ is associated with polynomials over finite field Z_2 and the degree of the polynomials is no more than seven. (Readers are referred to, for example, [Chi79, pp. 172, pp. 185 and LN97] for details of congruence classes modulo a polynomial)

Note that there are many degree-8 irreducible polynomials in $Z_2[x]$ [LN97, pp. 553] that can be chosen for constructing $GF(256)$. In the current implementation, the modulus is:

$$m(x) = x^8 + x^7 + x^6 + x^5 + x^4 + x^2 + 1,$$

where x, a formal symbol, is an indeterminant of the polynomial.

One way of thinking of the elements of GF(256) is to view them as *polynomials* in x with co-efficiencies in Z_2 of degree ≤ 7 and there are 256 (there are eight coefficiencies and therefore 2^8 possibilities) such polynomials. The operations of these elements are just like the operations of polynomials (for example, see [Chi79, pp. 125]. Addition and subtraction are done by bit-wise exclusive-or of the corresponding coefficiencies. To get the product again as a polynomial of degree ≤ 7, use the equation $x^8 = x^7 + x^6 + x^5 + x^4 + x^2 + 1$. Also, each element has a multiplicative inverse so that we can do division over the field as well. Notice that there are eight co-efficiencies of these polynomials and each of them is a bit. When putting these bits together and ordering from the leading co-efficient to the constant co-efficient, a binary number uniquely corresponding to a decimal integer in the

interval [0, 255] can be obtained. In that way, a field element can be uniquely associated with a character and vice versa.

We slightly modify (i.e., getting rid of the namespace) the GF(256) implementation from Wei Dai's Crypto library [Dai04] for our use.

4.2.2 Core Components

We have implemented ATIR as an ANSI C [Ker88] library with simple interfaces for secure and fault tolerant database access. In this section, we will describe the interfaces and protocols involved in an ATIR function call.

The ATIR APIs

Figure 4-3 shows the main APIs of ATIR client and server. On the client side, there are two major function calls. User programs can *invoke* ATIR client `tircd` by calling the function `tir_submit_query` with two parameters. `tir_sumit_query()` will perform three tasks: initialise the system, construct queries, and proceed the first attempt to send queries to the replicas.

```
Client:
int tir_submit_query(char *config_file, char *query_time);
int tir_get_result(char *result, char *result_time);

Server:
int tir_execute (int connfd, char *dbhost, char *dbusr, char
*dbpass);
```

Figure 4-3 The ATIR Library Main APIs

The `config_file` parameter is a pointer to the configuration file to initialise ATIR clients and contains two types of information: the user's ATIR requirements and authentication information. The user's requirements are used to generate queries and include the following: number of calculation records `numcalrecs`, undetected error rate e, range of valid data `data_range`, minimum number of correct replicas k, and maximum number of faulty replicas f. Note that `numcalrecs` is both the number of query elements sent to the server and the size of each block on the server side. The authentication information is used for establishing TCP connections with remote servers and includes the following: the IP addresses of ATIR servers, the

service port, database, table, and data item. The configuration file also contains the following specific information about the intended database, table, and data item with a pair of username and password to enable the servers to enforce access control. Currently, the library relies on the access control mechanisms provided by the servers.

The second parameter is a value-result argument [Ste98, pp. 65] which is a string pointer for storing the time taken to bootstrap the system from the configuration file, for preparing and sending queries. When the function returns, `query_time` stores the actual timing measurements.

Followed by calling `tir_submit_query` function, the user program should call the function `tir_get_result` to obtain results. The first parameter result is also a value-result argument which provides a storage space for storing the result reconstructed by the system. Similarly, the `result_time` parameter is used to return the time taken for reconstructing results. `tir_get_result()` conducts three tasks: i) continuing to establish the socket connections with the servers if they haven't been done, ii) sending queries and wait for the answers to return, and iii) reconstructing results and determine when to stop the reconstruction process.

On the server side, the main computation is triggered by the server daemon calling the function `tir_execute` with four parameters: the descriptor for an established socket, the database host name (in case the database server is separated from the ATIR server), and the username-password pair for accessing the database. In the current implementation, the ATIR server is multi-threaded. A new and separated thread is created to deal with the each client's ATIR request. A server assigns a new socket descriptor to each new connection which is stored in the parameter `connfd`. To ensure the service availability, three other parameters (i.e. `dbhost`, `dbusr`, `dbpass`) are also provided for authenticating legitimate users and identifying spurious ones.

Query Manager

This section describes how to prepare the query messages using the ATIR query protocol. The functionalities of the query manager include the following five tasks:

1) Generate a random record set for the server side computation;

2) Determine the order of the finite field;

3) Generate the query polynomials;

4) Set distinct evaluation points for the polynomials; and finally

5) Put all the information together to create the query elements for each replica.

Based upon the `identifier` passed by the client daemon and the `numcalrecs` setting, the query manager determines the record set by randomly selecting two integers `intendedindex` and `intendedblock` from the interval [1, `numcalrecs`]. The size (number of records) of each block is `numcalrecs`. The index of the first record `firstrec` is calculated as follows:

```
blocks = intendedblock - 1;
firstrec = identifier - intendedindex + 1 + blocks*numcalrecs;
```

The index of the last record `li` is calculated as follows.

```
blocks = numcalrecs - intendedblock + 1;
li = identifier - intendedindex + blocks * numcalrecs;
```

The selection should make sure that the index of the first and the last record of the sever side computation are within the interval [1, n], where n is the total number of records in the database. That is, `firstrec` should be no less than 1 and `lastindex` should be no greater than n.

Only transmitting the index of the first record rather than the indexes of all records significantly saves the communication cost. With the index of the first record, the server program derives other indexes on the server side. That is straightforward since the block size is fixed.

When the query protocol finishes, the query message for each replica is ready to be sent by the client daemon. The size of an index is different from the size of a query or an answer element. An index is a fixed-size (i.e., 8-bit) character whereas

the size of query and answer elements varies according to the size of the finite field that are chosen by the user.

Client Daemon and Result Manager

Logically, the client daemon is separated from the result manager. In the implementation, they physically coexist within one single program. That is because with the use of non-blocking I/O sockets, the connection operation returns immediately and informs the program that the operation cannot finish immediately. The client program can proceed to perform other operations and come back later.

Apart from coordinating the index resolver and query manager, *tircd*'s other tasks are to send queries and reconstruct results. Since ATIR clients connect to the servers through one-to-one communication channels, concurrent but separated socket connections are required between a client and each of the server to which it connects. We rely on TCP sockets to provide reliable communication.

There are two design alternatives for implementing multiple concurrent connections with servers: a multithreaded client with each thread dealing with a TCP connection with a server or a single-threaded client with non-blocking I/O processing. Our previous PIR/ATIR system implementation used the first approach with the Java programming language whereas our current implementation adopts the second approach using the C programming language. We presented the experimental results of the PIR/ATIR Java implementations in [YXB02a, YXB02b]. Compared with the second approach, the first one, however, is rather inefficient because of the use of blocking I/O and a polling model [Ste98 pp. 145]. The client daemon sits in a loop, which checks the availability of answers in the threads. This is often considered to be a waste of CPU time. Strictly speaking, it is not a proper approach for ATIR implementation because of its sequential checking method to poll the information of the threads. Before the system starts, it is unknown that which set of servers will first return answers.

Our current implementation uses the single-threaded approach to avoid the application level complication introduced by coordinating multiple concurrent threads and to reduce thread overhead. As shown in the next chapter, the C implementation of ATIR performs well. Indeed, non-blocking I/O processing has

been well documented to outperform thread processing. (See, for example, [Ste98, pp. 409], [KG02], and [Wel02]).

The client program follows an event-driven architecture by using the `select` function call to wait for answers to arrive or for timeout to be reached. We follow a similar approach used by Stevens in [Ste98, Chapter 15] to design the main logic of the non-blocking I/O processing. The client side sockets are all set to be non-blocking so that the system can effectively manage the socket connections and send/recv buffers. Generally speaking, blocking sockets will wait until the condition to be true while non-blocking sockets rely on the underlying system mechanisms to handle the concurrent events.

Specifically, we use non-blocking connect operations to attempt the first socket connection with each of the servers. Often, the first connection attempt will not be successful immediately. This may be caused by the slow response of servers and the delay of network transmissions. Since the system uses a non-blocking I/O, the socket will return immediately and report the error status (i.e. socket connection in process). Of course, the socket status will be checked later to check whether the connection has been established. By exploiting non-blocking I/O in our system, the impact caused by exceptional slow servers can be largely reduced and therefore the overall system performance may be improved. As demonstrated in our performance in the next chapter, this approach is can be very effective for dealing with slow or unstable network connections with remote servers.

Each replica is associated with a flag which can be in one of the statuses of the following set:

`{connecting, reading, done, failed}`.

After the first socket connection attempts, the socket status of all servers is set to be `connecting`. The first two statuses mean that the server is still interacting with the server. Once the connection is established, the client sends the corresponding query message immediately. Otherwise, the connection is failed and the server status is marked as failed.

Each socket has a pair of read/write descriptors that are used to examine whether the sockets are ready to read or write. When a non-blocking socket connection is established, the client sends the corresponding query to the respective server while

setting the read descriptor of this socket to be on and changing the server status to `reading`. The failure of the first non-blocking socket connection attempt does not mean that the server is not available. It may be the case that the server response or the network connection is slow. Therefore, the connection will be attempted later. In the meanwhile, the server status is set to be `connecting` and both read and write descriptor for this socket are set to be on.

Since we are using non-blocking sockets, there can be data available from *any* socket at any time. The data on TCP sockets is transmitted as segments which can arrive in one-go or separately. It is up to the underlying TCP mechanism to dynamically handle the transmission. Therefore, it is mandatory to ensure that the client keeps checking the socket until no more data is available to read. For a large trunk of data that are transmitted in several TCP segments, the underlying socket handles the actual data amount transmitted in each segment despite the total number of bytes sent out by the server. Each time, a newly arrived message segment is appended to the end of the current answer buffer. Once the number of ready sockets exceeds the threshold limit, the reconstruction function will start to reconstruct results.

Server Daemon, View Manager, Answer Manager

The server daemon is implemented as pre-threaded servers to handle multiple clients' concurrent connections due to its good performance over non pre-threaded servers [Ste98]. By creating a pool of threaded when a server starts, it reduces the time taken to deal with each connection as the server can just reuse the existing threads without creating new ones. `mutex` is used to control concurrent access to critical regions and variables, for example, the `accept()` function. Once a new thread is created to handle the client's connection, the program control is handed over to the function `tirexecute()`, which has a loop to produce the answer for each block repeatedly.

For each block, the view manager does three tasks. First, it constructs a SQL statement based on `numcalrec` and `firstrec` parameters obtained from the received query structure. It then proceeds to execute the SQL statement and transform the SQL results into a computable format.

4.2.3 Message Formats

With the use of non-blocking I/O operations, however, we do need to pay some extra attention to socket buffer management. In our case, we have paid particular attention to the format of the query and answer messages to tune for better communication complexity. Figure 4-4 and Figure 4-5 shows the message formats in our current implementation. There are two types of messages exchanged over the communication channels in ATIR. A message from the client to servers is called a query and a message on the reverse direction is an answer. The major content of query messages is query elements whereas the major content of answer messages is answer elements. In general, we refer to the major content as elements.

These formats only specify the maximum communication capability of ATIR messages. Each message contains two parts: a fixed-length header and a non-fixed length data. The length of headers is fixed no matter how much data items to be retrieved. The actual size of messages transmitted over the network varies mainly due to the size of contents (i.e., actual queries and answers). The following specifies the meanings of each part.

- **cid**: an 2-byte integer which specifies the client id (currently unused)
- **gforder**: set the order of the finite field
- **numcalrec**: number of records in a block
- **firstrec**: the index of the first record
- **dbinfo**: the name of the intended database, table, username and password
- **content**: query elements

An answer message contains two sections: `len` and `content`. `len` is the total number of elements in each block and $\texttt{elen}_j$, where $j = 1, 2, \ldots, \texttt{flen}$ and `flen` is the number of fields in a block. `len` and $\texttt{elen}_j$ has the following relationship:

$$\texttt{len} = \sum_{j=1,2,\ldots,flen} elen_j .$$

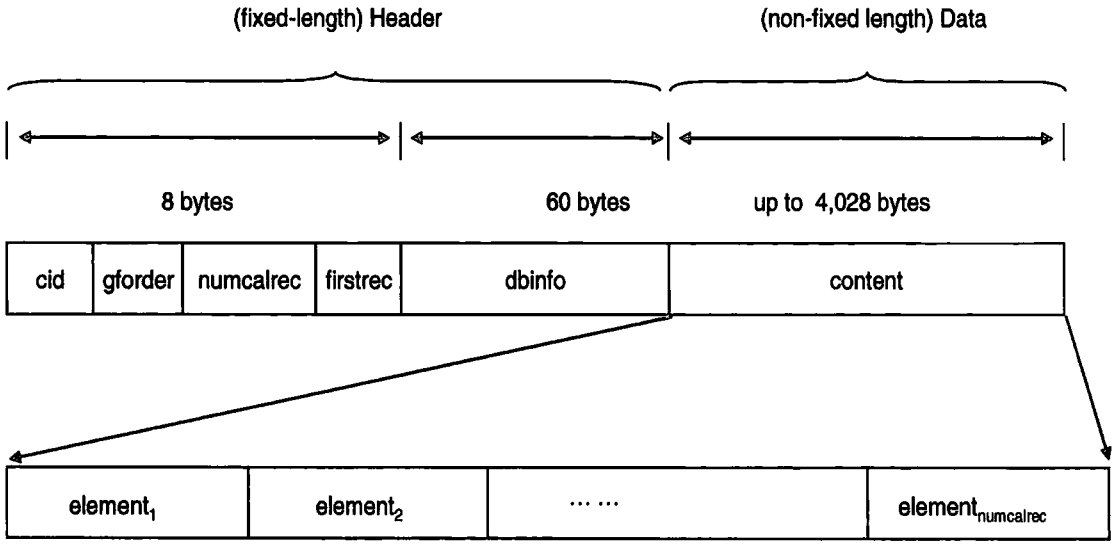


Figure 4-4 Message Format for Queries

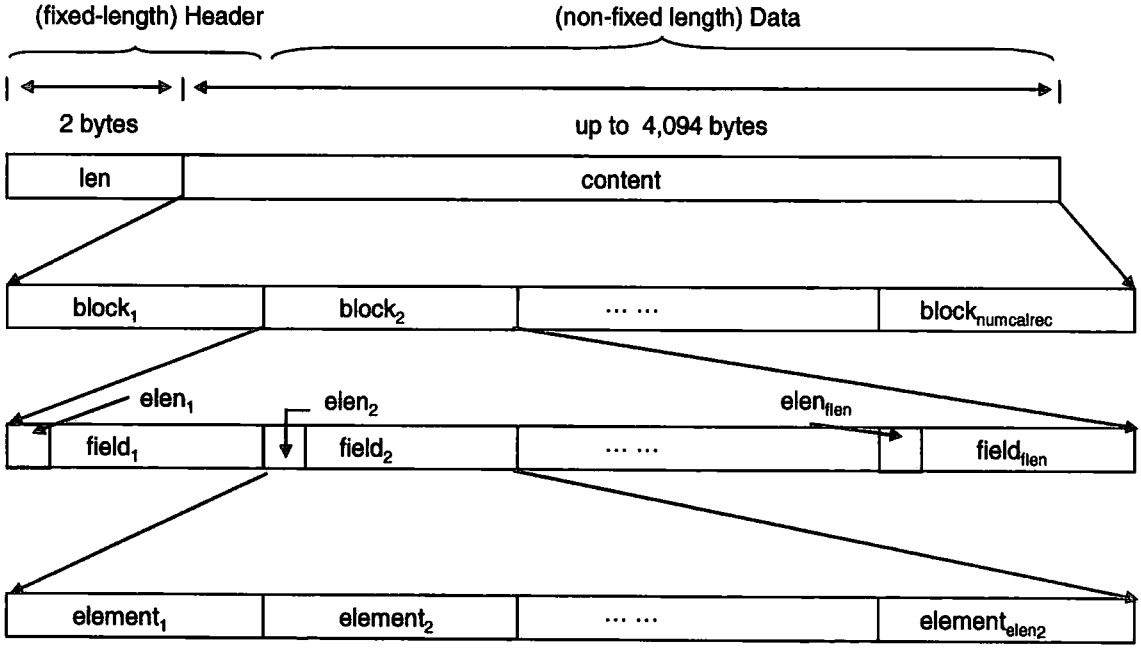


Figure 4-5 Message Format for Answers

4.2.4 Optimizations

This section describes three optimisations used in the PIR/ATIR C implementation.

Structure optimisations

The first optimisation only applies to the PIR implementation and reduces the communication cost by half, comparing to that of the ATIR implementation. This reduction is significant for PIR requests which require high privacy protection.

In PIR, the elements in query and answer messages are 1-byte integers/characters, which are enough to accommodate all the characters that could appear on the server side computation. Therefore, these elements are sufficient to cover all the field elements that are required for PIR computations. However, ATIR has to use two-byte integers due to the fault tolerance requirement. Excluding the constant communication overheads imposed by the message headers, this technique reduces the communication by half.

Probabilistic Reconstruction for ATIR

This optimisation speeds up the reconstruction process of ATIR. Instead of waiting for the finish of the deterministic reconstruction, the probabilistic reconstruction stops when the first valid result is obtained. With the use of the pATIR, the reconstruction time can be reduced significantly in the normal situation. For example, with the use of three servers, the reconstruction time of pATIR is 1/3 of that of dATIR in normal circumstances.

MySQL Initialisation

In the current implementation, MySQL server side initialisation performs once before the block processing. This is an alternative to initialise MySQL connections for each block. This saving increases as the number of blocks grows. For example, when `numtok` = 100, i.e., there are 100 blocks, the total processing time is reduced from 0.277 second to 0.246 second. In this case, the optimisation saves 11% of the total processing time of ATIR.

4.3 Implementation Issues

4.3.1 Circumventing the Index Knowledge Assumption

There are several inherent stumbling blocks that preclude the practical ATIR implementations. As noted by an early paper [CGN97], all known PIR schemes require the user to know exactly the *physical index* of the intended record in a database. This assumption suffers from the following limitations: i) the physical index of records changes all the time, and it is difficult to keep users updated with the latest changes; ii) the implementation of indexing mechanisms varies from system to system. In most of the cases, user applications cannot directly access this information. Often, this information is transparent to the user applications. The implementation of ATIR schemes also faces this problem.

We can relax this assumption by associating each entry in the database with a unique numerical identifier, and introducing an identifier resolution protocol to extend the ATIR service from hiding the index of an intended data item to hiding the keyword of the data item.

When the user supplies the keyword of an entity, the ATIR service will query a directory service with the keyword and returns two pieces of information to the user: the identifier of the keyword and the total number of identifiers in the directory. Currently, our implementation only supports an one-to-one mapping relationship between a keyword and an identifier.

This adaptation removes the index knowledge assumption and paves the way for integrating ATIR service in real applications. In practice, the physical index of the data item in databases can be substituted with the identifier information. An identifier is a label that identifies a person or an entity. Examples of identifiers include primary keys of databases, names of objects in an object-orientation programming, and the newly proposed permanent identifier for public key certificates [PG04]. Identifiers are usually required to be *unique* to ensure the one-to-one mapping relationship between an identifier and a subject entity.

We assume that this mapping relationship is published correctly by service providers and stored in a trusted public directory. By searching the public directory using a subject name, we can resolve the identifier of an entity and then use it to retrieve the corresponding data item. This is through the use of the Identifier

Resolution Protocol that returns a unique identifier corresponding to the inputted keyword.

4.3.2 Full-length Processing versus View Processing

The server side computation of PIR schemes can be classified as: pre-processing and online processing. Both computations process the *entire* database on each individual server. We call it full-length (pre-/online) processing. Only one of them will be chosen for any specific PIR scheme. The pre-processing aims to reduce the cost of online processing by changing the data to a specific format. This approach is used by [IS03], which reduces the cost of online processing to a constant. In terms of online processing, this is optimal although it is at the cost of full-length pre-processing. Without pre-processing, online processing needs to compute over an entire database.

The full-length processing over the entire database guarantees the perfect privacy property in PIR schemes, i.e., each record has an equal probability of being the one a user wants. For example, if the processing is over a 100-record database, each record has 1/100 chance being the one wanted. Although the full-length processing is secure, it is costly and not flexible. Generally speaking, the level of privacy protection is proportional to the number of records involved in the server side computation. In other words, the chance of successful privacy violation by a server is inversely proportional to the number of records involved in the server side computation. This is a trade-off between computation cost and privacy. A better strategy is to let the user decide the amount of time for processing, and correspondingly, the level of privacy protection an ATIR service can provide. For example, by specifying the number of records the user wishes to compute over, a user can choose the level of privacy protection s/he can get.

4.4 Summary

This chapter first presents the architectural design of our ATIR system and the character-string database model used in the ATIR system implementation. We then describe the design details of the major components and present the message formats used in the ATIR system. A number of optimisation techniques used in the implementation of ATIR are briefly described. That is followed by some discussion on the important implementation issues of the ATIR system.

Chapter 5 Empirical Evaluation

This chapter presents the experimental studies of ATIR systems. We first derive an analytic model for the performance of an ATIR service and then derive performance model parameters. We validate the model by showing that it accurately predicts the performance results gathered in the experimental studies. The model can also be used to predict the performance of the system in different settings.

We further examine the impacts of varying the major parameter, such as view sizes and result sizes, on the Total Time for Processing (TTP) of ATIR in fault-free situations. To understand the source of performance bottleneck of both services, we also examine the contributions of each component to the TTP.

Finally, we investigate the behaviours of the ATIR system in the presence of simulated faults. We focus on the impacts of these faults to the TTPs of ATIR and discuss the implications of the experimental results. This chapter also compares the performance of ATIR with that of PIR and the downloading solution.

5.1 Performance Models

This section investigates the major factors that make an impact on the ATIR system performance. We present a complete analytic model for the ATIR implementation to analyse the costs imposed by various operations in the system. Performance models are useful for understanding and explaining performance results. We can use these models for identifying the major sources of performance bottlenecks and verify the importance of theoretical concerns. These models are also useful for predicting the performance of the ATIR system in a different setting. The analytic model consists of a number of component models where each of them models an individual function of the system.

5.1.1 Preliminaries

The following notations are used in the performance models throughout this chapter. A symbol with f as its subscript means that it is a fixed cost incurred for performing an operation, that is, this cost is independent of the input of a function. A symbol

with a as its subscript means that it is an additional cost incurred per unit. A unit can be a byte, an element, or a record. An element is either a single-byte integer or a two-byte integer. An element is used in both computational and storage contexts. The element size, denoted by *es*, is defined as the number of bytes in an element.

A result is the parts of a record that a user wants to retrieve. The result size, denoted by *rs*, is defined as the original number of elements in a result. By trimming off the empty spaces at the end of a result, an optimised result size, denoted by *ors*, is obtained.

A query message contains a fixed-sized query header and query elements. The number of query elements in the message is defined as query size, denoted by *qs*. The query message size is the total number of bytes in a query message.

An answer message consists of a fixed-sized answer header and answer elements. The number of answer elements in the message is defined as answer size, denoted by *as*. The answer message size is the total number of bytes in an answer message.

The size of a message, denoted by *mess*, is the total number of bytes in a query and an answer message.

A view is a collective representation of the data sets involved in the server side computation. A view size, denoted by *vs*, is defined as the total number of records involved. Table 5-1 offers a list of the notations for the variables described.

Table 5-1 Notations for Performance Models

Variables	Name	Unit	Description
es	element size	bytes	number of bytes in an element
qs	query size	element	number of elements in a query, exclusive of a fixed-sized query header
as	answer size	element	number of elements in an answer, exclusive of a fixed-sized answer header
rs	result size	element	number of elements in a result
mess	message size	byte	number of bytes in a message
vs	view size	record	number of records in a view
ors	optimised result size	element	number of elements in a result after trimming off the spaces at the end

In the ATIR implementation, these variables have the following relationships:

$$as = qs \times rs$$

$$mess = (qs + as) \times es$$

$$vs = (qs)^2$$

5.1.2 Performance Modelling

The overall performance of an ATIR service is characterised by TTP - the time taken from the system starting to process an ATIR configuration file until the system obtaining a result. TTP can be divided into three processing time as follows: client side, server side, and communication.

On the client side, the major components of TTP include: system initialisation, query message preparation, and result reconstruction/verification. On the server side, the major components of TTP are: view creation, and answer message preparation. To help our presentation, let us first give a list of the short forms for timing variables and their corresponding meanings.

TIS: time taken to initialise the system. (a constant cost)

TPQ: time taken to prepare query messages, including headers.

TRV: time taken to reconstruct and verify results.

TCV: time taken to create views.

TPA: time taken to prepare answer messages, including headers.

TTQM: time taken to transmit a query message.

TTAM: time taken to transmit an answer message.

We further introduce the following timing variables to represent the sums of the above timing components:

TSP: server processing time.

TCOMM: communication time of an ATIR service.

TSR: server response time.

They have the following relationships:

$$TSP = TCV + TPA$$

$$TCOMM = TTQM + TTAM$$

The client also spends time to perform the following regular tasks: establishing and closing TCP connections, method invocations, memory management, establishing and releasing MySQL server connections, and garbage collections. We refer them as the time taken to do miscellaneous tasks, denoted by TMIS.

Figure 5-1 shows a timing diagram which illustrates the relationships among these variables. (For clarity, TMIS is not presented in the diagram.) The shadow boxes represent the participants of an ATIR service. Each participant is associated with an execution line from top to bottom. To simplify the presentation, the diagram assumes that the client concurrently sends the query messages to the servers respectively and the servers receive them at the same point along the time line. A similar situation is also assumed for the answer messages. In reality, many non-system factors may have various degrees of impacts on message arrivals. Network load, system load, and system management tasks are the typical factors.

The TTP of ATIR is given by the following formula:

$$TTP = TIS + TPQ + TCOMM + TCV + TPA + TRV$$

In the remaining presentation of this section, we shall focus on these components: TPQ, TPA, TCOMM, TCV, TRV, and finally TTP. The communication cost in ATIR is quantified through two models: a message model (TCOMM-1) and a communication cost model (TCOMM-2). Message sizes and actual communication costs are the major concerns in PIR and ATIR research. TCOMM-1 represents the bandwidth consumption of ATIR whereas TCOMM-2 characterises the performance overhead of message transmission.

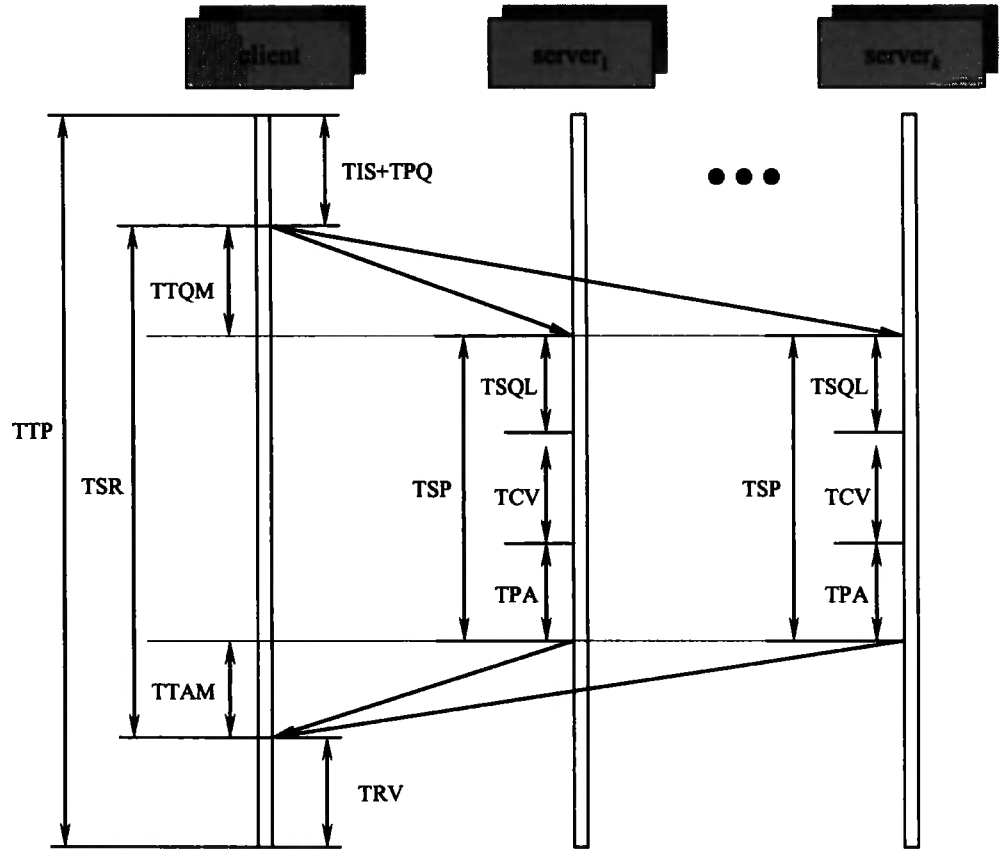


Figure 5-1 A Timing Diagram for ATIR Services

Message Preparation (TPQ and TPA)

We first derive a model for calculating TPQ. The computation is modelled as a linear function of query size as follows.

$$\mathrm{TPQ}(qs) = \mathrm{QM_f} + \mathrm{QM_a} \times qs$$

QM_f is a fixed cost of computing a query which contains zero query elements and is measured in microseconds. QM_a is an additional cost of computing each query element and is measured in microseconds per element.

We then derive an answer model for calculating TPA which is modelled as a linear function of view size and result size.

$$TPA(vs, rs) = TPA_f + TPA_a \times vs \times rs$$

where vs is the view size and rs is the result size.

TPA_f is a fixed cost of computing an answer message which contains zero elements and is measured in microseconds. TPA_a is an additional cost of computing

with each element and is measured in microseconds per element. $vs \times rs$ is the number of elements in an answer.

Communication Model 1: Message Model (TCOMM-1)

We first derive a message model which calculates message sizes based on query sizes, result sizes, and element sizes. Message sizes correspond to the bandwidth consumption of ATIR services. Since message sizes are independent of any specific network infrastructure, it is useful to analysis them when communication cost is discussed.

Between a client and a server, the total message size transmitted, denoted by *tms*, is the sum of the size of a query message and an answer message. We have the following formula which calculates the total number of elements exchanged between a client and a server:

$$tms\ (qs, ors, es) = qh + ah + qs \times (es + ors \times es)$$

where qh is the size of a query message and ah is the size of an answer message.

Alternatively, the above formula can be revised as a parameter of vs as follows:

$$tms\ (qs, ors, es) = qh + ah + vs^{1/2} \times (es + ors \times es)$$

In PIR, each element is represented by a one-byte integer and thus the *tms* of PIR is also the number of bytes exchanged between a client and a server. In ATIR, each element is represented by a two-byte integer and thus the number of bytes exchanged between a client and a server doubles the above *tms*.

Communication Model 2: Communication Cost Model (TCOMM-2)

We now derive the second communication model (TCOMM-2) to model the actual communication cost of transmitting the messages in the ATIR system. This second communication model is based on the first communication model (TCOMM-1) because it relies on the first model to calculate the size of a message. The second model separates the host processing overhead from the actual network communication overhead. The separation enables us to predict the performance of the system in a different network setting, such as the Internet. Since our implementation uses TCP connections for message transmission, our discussion focuses on TCP.

The communication model of an ATIR system consists of two parts: the model of host processing and the model of network processing. The former model aims to model host processing time whereas the latter models network communication time.

The communication costs in both models are modelled as a linear function of the variable *tms*. For the host-processing model, there are two parameters: a fixed cost H_f of processing zero byte data and an additional cost H_a of processing an extra byte. For the network-processing model, there are also two parameters: a fixed cost N_f of transmitting zero byte data and an additional cost N_a of transmitting one-byte data. H_f and N_f are measured in microseconds whereas H_a and N_a are measured in microseconds per byte. Therefore, TCOMM is calculated as follows:

$$\text{TCOMM}\ (tms) = H_f + 2 \times H_a \times (tms) + N_f + N_a \times (tms)$$

View Creation (TCV)

This section presents a model for computing *TCV*, i.e. the time taken to get records from databases and transform them into a view. Again, the model consists of two components: a fixed overhead TCV_f of performing the transformation and an additional cost TCV_a of transforming an element. TCV_f is measured in microseconds and TCV_a is measured in microseconds per element. The input to the model is view size and result size. *TCV* is modelled as follows.

$$TCV\ (vs, rs) = TCV_f + TCV_a \times vs \times rs$$

Result Reconstruction and Verification (TRV)

This section describes a model for computing TRV, i.e., the time taken to reconstruct a result. The TRV model consists of two components: a fixed cost TRV_f of reconstructing and verifying a record and an additional cost TRV_a of reconstructing and verifying an element. The model is as follows:

$$TRV\ (ors) = TRV_f + TRV_a \times ors$$

TRV_f is measured in microseconds and TRV_a is measured in microseconds per element. Each reconstructed element is subject to verification. Therefore, TRV can be further divided into two sub-components: the time taken to reconstruct a result and the time taken to verify a result.

In theory, it is necessary to separate TVER from TREC because there is one of the major differences between PIR and ATIR. However, as we shall show in the experimental studies, the separation is not necessary because of the cost of verifying a result in TTP is effectively negligible. Therefore, we only present one combined model for the result reconstruction and verification.

Total Processing Time (TTP)

Putting all these models together, we have the following formula for computing the TTP of an ATIR service.

$$TTP_{atir}(es, vs, rs, ors) = TIS + TPQ(qs) + TCOMM(qs, ors, es) + TCV(vs, rs) + TPA(vs, rs) + TRV(ors) + TMIS$$

5.2 Experimental Setting

Unless otherwise stated, the experimental settings described in this section apply to all experiments presented in this chapter.

5.2.1 Experimental Objectives

In this thesis, there are four goals of conducting our experimental studies:

- Deriving the analytic model of the performance of an ATIR service;
- Quantify the cost of providing privacy protection;
- Investigating the cost of dealing with attacks in ATIR;
- Comparing the experimental results of ATIR with SQL queries and PIR.

5.2.2 Experimental Environment

Our experiments use a set of identical DELL machines with the following specifications: Dell Precision 650 workstation with Dual Intel Xeon 3.06GHz hyper-threaded, 1 GB RAM, 36GB Fujitsu MAS3367NP SCSI hard drive, and 3COM 3c905C NIC. They are interconnected through a Cisco switch which has the following specifications: Cisco Catalyst 2924 (model: WS-C2924C-XL-A). The switch has twenty-two 10/100BaseTX ports and two 100BaseFX ports.

All machines run Redhat Linux release 9 (Shrike) and the kernel version is 2.4.26. The kernel is compiled with smp[1] support. The specifications of the major software packages are described as follows. The version of MySQL is 11.18 with distribution 3.23.58 for redhat-linux-gnu (i386). All the programs used in the experiments are compiled with GCC 3.2.2.

Three machines are used as servers in all ATIR experiments. Two machines are used as servers in all PIR experiments whereas one machine is used as a server to perform SQL queries. One client machine is used in all the experiments presented in this chapter.

The database (precisely the database table) used in the experiments contains over 44,000 records where each record contains 880 bytes. Apart from otherwise specified, the default size of a result in the experiments is 10 bytes.

Unless stated otherwise, the finite field for ATIR computation is GF(257) and the finite field for PIR computation is GF(256).

Again, unless stated explicitly, we use a simple (single factor) linear regression method to compute the model parameters in all the experimental studies.

Due to the resource restriction, the network is not an entirely private closed network. For system administration purposes, there are regular synchronisation jobs running on the machines which are beyond our control. That means there may be a small amount of bursting traffic in the network which may have unpredictable impacts on system measurements. However, as any random factors, such as tasks running by operating systems, we try to minimize such unpredictable impacts through conducing repeated experiments, excluding outliers, and using statistical methods to justify the accuracy and validity of the experimental results.

It is, however, observed that some outside factors do have an impact on the performance of our experiments. For example, the first execution of any program often experiences exceedingly high (twice or three times more than usual) performance overhead. We believe it is due to the context switching management performed by the operating system. Another such outside impact occurs with the use of MySQL database services, on top of which the system is built. It is observed that

[1] smp: shared memory multiprocessor

the time taken to establish a connection with a MySQL server at the first time is two or three times of that in the subsequent establishment. The exact cause of this phenomenon is not clear. Since this does have a direct impact on our system performance, we repeat all experiments eleven times and eliminate the first result.

5.2.3 Primitive Component Model Parameters

Concerns of Deriving the Communication Cost Model

Host processing and network processing are two major sources of TCP performance overheads. Previous studies (e.g. [CJRS89]) have shown that host processing can impose noticeable overheads on the overall TCP performance. Here, a host is defined as the computers at the end points of a network. Host processing is mainly software overheads and includes, for example, the costs of running TCP program on the computers (i.e., at the application layer), and the costs of moving bytes in the memory (i.e. between user address space and system address space and between network interfaces to system address space).

A network encompasses both intermediate processing nodes (e.g. routers) and communication links. Network processing overheads include the costs that arise from processing the packets on intermediate notes and communication links. Separating host processing from network processing gives us a better insight on the real source of the communication costs of TCP. Furthermore, the separation can help us to predict the performance in a different network environment with other settings.

As similarly assumed in other communication experiments (e.g. [KR01]), we make a number of assumptions to simplify the measuring task. Using these assumptions keeps the thesis in focus. With respect to the communication model, our goal in this thesis is to model the communication cost of the system in action. It is not our goal to develop a generic methodology to model the communication of the TCP/IP protocols. In a network environment, the communication latency would depend on several parameters, such as retransmission, flow control policies of the TCP protocol, buffer sizes of senders and receivers, the communication distance, bandwidth availability, traffic intensity, and the number of intermediate hops in the network communication path.

We assume that all the communication channels in our system are dedicated and the bandwidth availability is the same. We also assume that the network has no congestion, and packages are neither lost nor corrupted so that the TCP protocol will not retransmit packets. From the statistics data collected, this assumption seems to be valid for our experiments due to the fact that it is a closed and dedicated network environment.

Host Processing Time

We need to run two batches of experiments because our communication model (TCOMM-2) separates the host processing from network communication. Host Processing Time (HPT) is the time taken to process a message on a single computer. Network Communication Time (NCT) is the time taken to transmit a message by a network from one computer to another computer. Round Trip Time (RTT) is the double of the sum of HPT and NCT. It should be noted that most RTT measurements in the literatures do not distinguish HPT and NCT from RTT. In our studies, however, the separation provides us with a way to predict the communication overhead that may exhibit by the system in a different network setting.

The first batch of experiments aims to measure HPT. That is, the client and the server reside on the same computer. Therefore, the measured host processing overhead is the sum of the following two costs: the cost of segmenting and copying the messages from the client (program) address space to the system address space and the cost of reassembling and copying the messages from the system address space back to the client address space. Essentially, the host processing cost is the cost of going through the TCP protocol stacks (TCP/IP).

To measure HPT, we resort to the loopback address (or interface) [Ste94]. A loopback address specifically refers to the local host IP address 127.0.0.1. Any messages send to this address will be routed back to the originated source. The roundtrip time obtained through a loopback address is, therefore, the cost incurred by the software processing on a single computer. Table 5-2 shows the measured roundtrip and one-way HPT. One-way HPT is half of the roundtrip HPT. Based on the measured data, we compute the parameters H_f and H_a, which are shown in Table 5-3. The parameter is calculated based on the one-way HPT.

Figure 5-2 plots the measured and predicted one-way HPT against various message sizes. The predicated values match with the measured values with a high co-efficient of determination (98.37%). That means the prediction is accurate.

Table 5-2 Measured Host Processing Time (μs)

Message size (bytes)	Roundtrip (μs)	One-way (μs)	Std. Dev.
100	29.60	14.8	2%
200	30.40	15.2	5%
300	31.10	15.55	2%
400	30.90	15.45	5%
500	31.20	15.6	3%
600	31.50	15.75	4%
700	32.20	16.1	2%
800	32.90	16.45	4%
900	33.80	16.9	2%
1000	33.60	16.8	3%
1100	34.30	17.15	4%
1200	34.80	17.4	2%
1300	35.30	17.65	3%
1400	35.30	17.65	5%
1448	35.80	17.9	3%

Table 5-3 Parameters of One-way HPT

Parameter	Value	Description
H_f	14.629 μs	A fixed overhead of transmitting messages of any size through the loopback interface
H_a	0.002252μs/byte	An additional cost of transmitting a single byte through the loopback interface

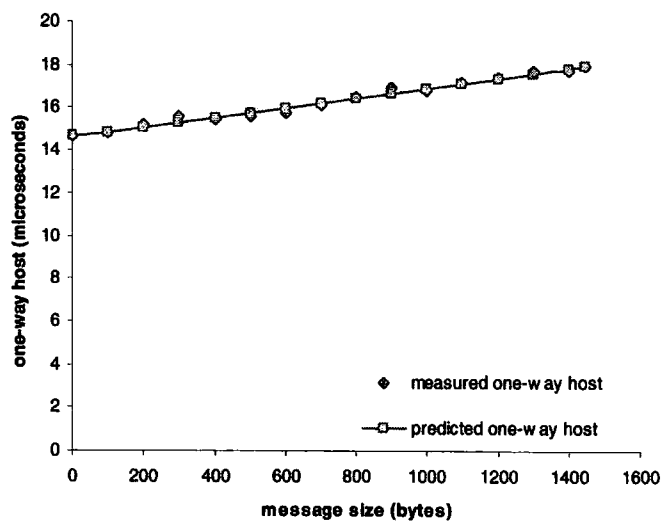


Figure 5-2 Measured and Predicted One-way Host Processing

When the system is deployed in a different network environment, the potential impact of the network environment becomes a dominant factor on the stability of TTP. If the same set of machines is used for the new deployment, the HPTs will remain the same since they are independent of the underlying network infrastructure.

Network Communication Time

The second batch of experiments aims to investigate the relationship between NCT and message sizes. NCT is obtained through subtracting HPT from RTT. Table 5-4 shows the measured RTTs and calculated one-way NCT obtained through experimenting with two separated computers. The second column (i.e., RTT) is the time taken to send the messages back and forth between the computers. The third column (i.e., Std. Dev.) is the standard deviation, which is the variance of the ten observations from the averaged RTT. The last column is the one-way NCT which is the time taken to transmit the messages from the client to the server, which is half of the difference between the RTT and four times of the corresponding one-way HPT. Based on the measured data, we compute the parameters N_f and N_a, which are shown in Table 5-5.

Table 5-4 Measured Roundtrip Time and One-way Communication Time

Message size (bytes)	RTT (μs)	Std. Dev.	One-way NCT (μs)
100	497	1.64%	219
200	556	0.66%	248
300	629	0.25%	283
400	612	0.47%	275
500	690	0.49%	314
600	687	0.36%	312
700	746	0.26%	341
800	788	0.42%	361
900	837	0.70%	385
1000	875	0.66%	404
1100	938	0.46%	435
1200	982	0.40%	456
1300	999	0.00%	464
1400	1037.4	0.25%	483
1448	1047.7	0.31%	488

Table 5-5 Parameters of One-way Network Communication Time

Parameter	Value	Description
N_f	205.36 μs	A fixed overhead of transmitting messages of any size through the network (one-byte)
N_a	0.19978 μs/byte	An additional cost of transmitting one-byte through the network (one-byte)

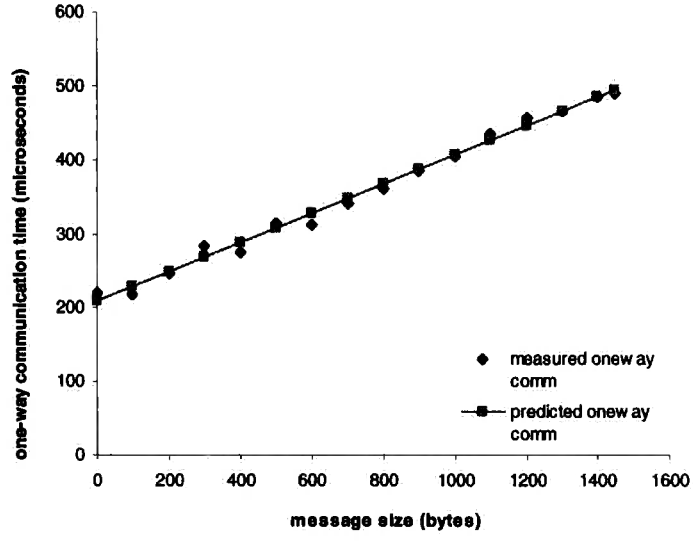


Figure 5-3 Measured and Predicted One-way Communication

Table 5-6 Protocol Constants

Name	Value	Description
TSQL-INIT	238 μs	time taken to establish a connection with a local MySQL server
TIS (ATIR)	74 μs	time taken to initialise the ATIR system

Figure 5-3 plots the one-way NCT between two computers in our network. The predicted values match with the measured values with a high co-efficient of determination (99.09%)

Constants

Table 5-6 shows the protocols constants for all the experiments. These values are used to produce to predict TTPs in the subsequent sections.

5.3 ATIR Experiments in a Fault-free Environment

This section first derives the component parameters for the ATIR models and then investigates the impacts of varying view size, result size, undetected error rate on the TTP of ATIR and the proportions of each component in the TTP of ATIR. We also investigate the impact of using a deterministic ATIR (dATIR) to the system performance. Unless mentioned explicitly, an ATIR scheme refers to a probabilistic ATIR scheme. We further present the experimental results of ATIR performance in the presence of various simulated faults.

5.3.1 ATIR Component Model Parameters

Message Preparation

We use a simple linear regression model to find out the parameters TPQ_f and TPQ_a in the query model and TPA_f and TPA_a in the answer model. Table 5-7 shows the parameters obtained through using the regression (least square) method.

To verify the query model, we run ten independent experiments varying the query size from 10 elements to 100 elements which are then used to compute from 100 to 10,000 records on the server side. Each experiment involved a different-sized query message. In each series, each experiment is consecutively repeated eleven times.

Figure 5-4 shows the measured and predicted TPQ as the query sizes increase. Each diamond point is the average value of five runs. Apart from the first series of experiments, the TPQ time for the overall experiments is fairly stable. The standard deviation of the averaged values of the first experiment is 7% whereas that of the rest of the experiments is below 5%. The model is also accurate. The co-efficient of determination is 97.5%. The high co-efficient of determination means that the model matches with the measured data.

Table 5-7 Message Preparation Parameters (ATIR)

Parameter	Value	Description
TPQ_f	9.09194 µs	A fixed overhead of computing query messages of any size
TPQ_a	0.7231 µs/element	Additional cost per element in a query message
TPA_f	2 µs	A fixed overhead of computing answer messages of any size
TPA_a	0.9246 µs/element	Additional cost per element in an answer message

To verify the answer model, we also log *TPA* in the above experiments on the server side. Corresponding to the query size from 10 to 100 elements, the view size ranges from 100 to 10,000 records. Figure 5-5 plots the measured and predicted *TPA* against the view sizes. The co-efficient of determination of the experiments is over 99.99%. This means that our model is accurate.

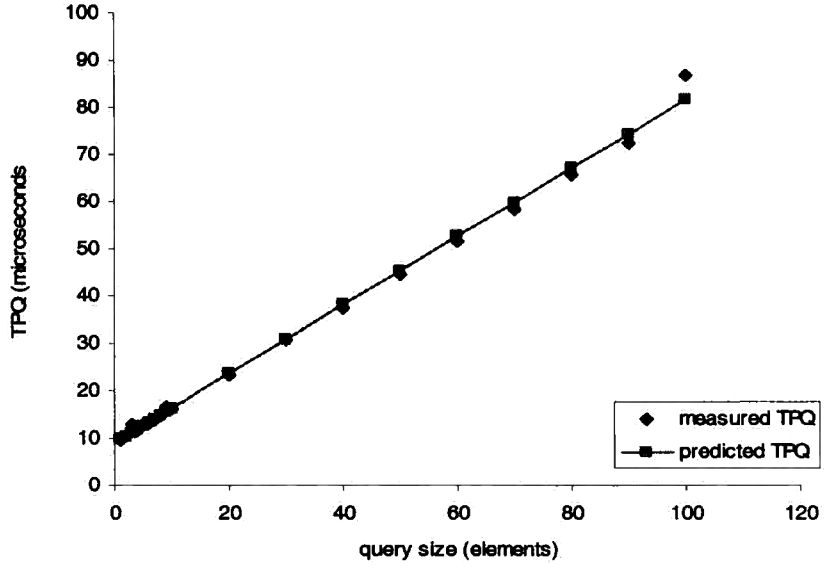


Figure 5-4 Measured and Predicted TPQ vs. Query Sizes

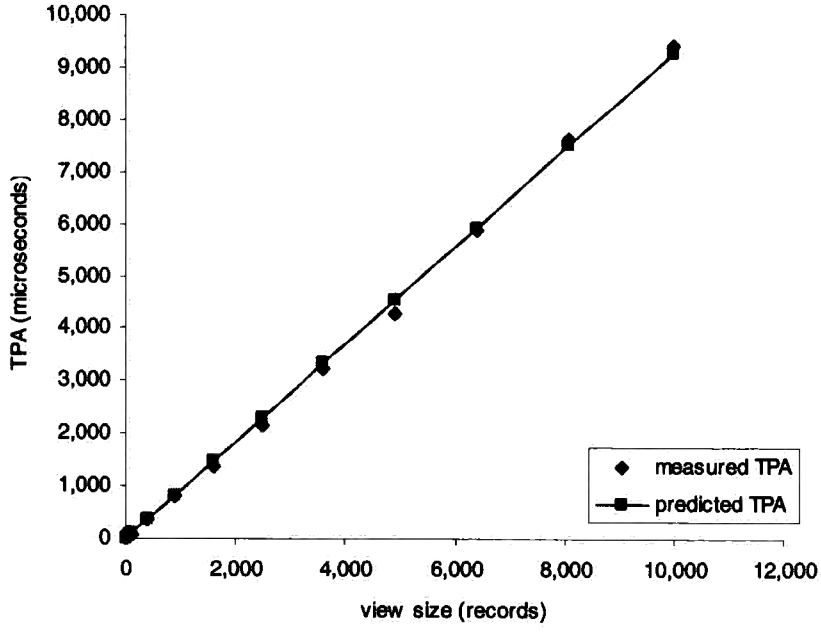


Figure 5-5 Measured and Predicted TPA vs. View Sizes

Communication

As explained in Chapter 4, it is sufficient and necessary for the ATIR implementation to use a 2-byte integer to represent an element. Table 5-8 shows the relationship among message sizes, query sizes, view sizes and the total communication time for transmitting the messages. The original result size is 10 bytes and the optimised result size is 6 bytes. Since the answer size is determined by the optimised result size, we simply assume that the result size is 6 bytes per block.

In order to understand the table, let us take the first row as an example to explain the meanings of these figures. In this row, an APIR query computes over 100 records. Therefore, the query contains 10 elements with 2 bytes each and the answer contains 10 blocks. The query message size is 120 bytes, which is the sum of a 100-byte query header and 20-byte query elements. The answer message size is 122, which is the sum of a 2-byte header and 120-byte (i.e., 12 bytes/block × 10 blocks) answer elements. Using the communication parameters derived in Section 5.2.3, we can obtain the total communication time as follows.

$$2*14.629 + 205.36 + 242 \times (2*0.002252 + 0.19978) = 284.054\ \mu s$$

It is important to note that the total message size of ATIR is solely determined by the view size. The deployment in a different network environment will only have an impact on the total communication time, not the total message size.

View Creation

To verify the model, we use the linear regression method to calculate the parameters TCV_f and TCV_a which are shown in Table 5-9. Figure 5-6 plots the predicted and measured costs of TCV against view sizes. The model is accurate because the co-efficiency of determination is 99.97%.

Table 5-8 Query Sizes, View Sizes, Message Sizes and Communication Time of ATIR

Query Size (elements)	View Size (records)	Bsent (bytes)	Brecv (bytes)	Total Message Size (bytes)	Total Comm. Time (μs)
10	100	120	122	242	284
20	400	140	242	382	313
30	900	160	362	522	341
40	1600	180	482	662	370
50	2500	200	602	802	398
60	3600	220	722	942	427
70	4900	240	842	1082	456
80	6400	260	962	1222	484
90	8100	280	1082	1362	513
100	10000	300	1202	1502	541

Table 5-9 Parameters of TCV

Parameter	Value	Description
TCV_f	153 μs	A fixed overhead of performing DB transformation operations
TCV_a	13.573 μs/record	Additional cost per record of performing DB transformation operations

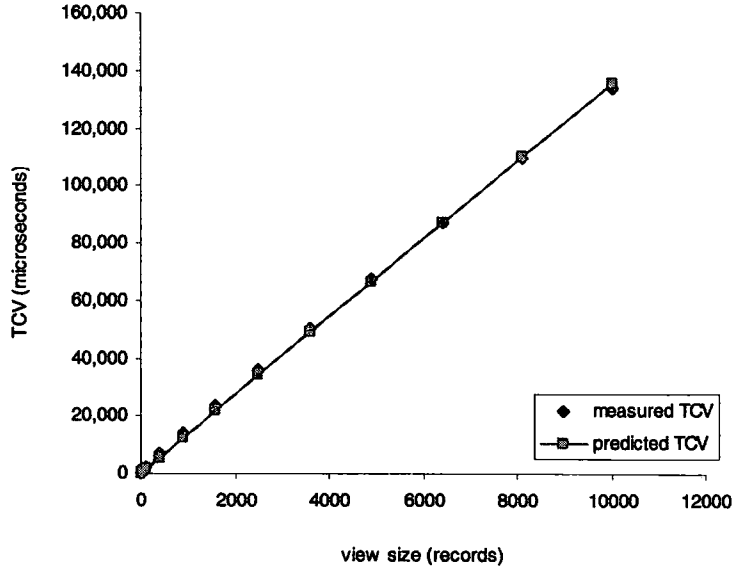


Figure 5-6 Measured and Predicated TCV vs. View Size

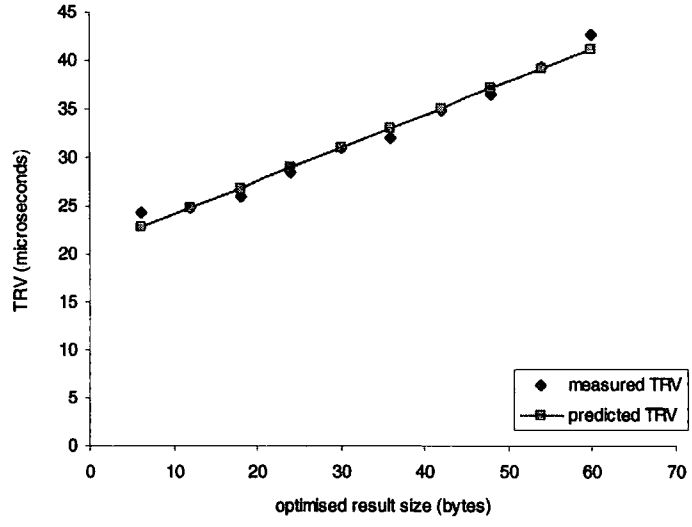


Figure 5-7 Measured and Predicted TRV vs. Optimised Result Sizes

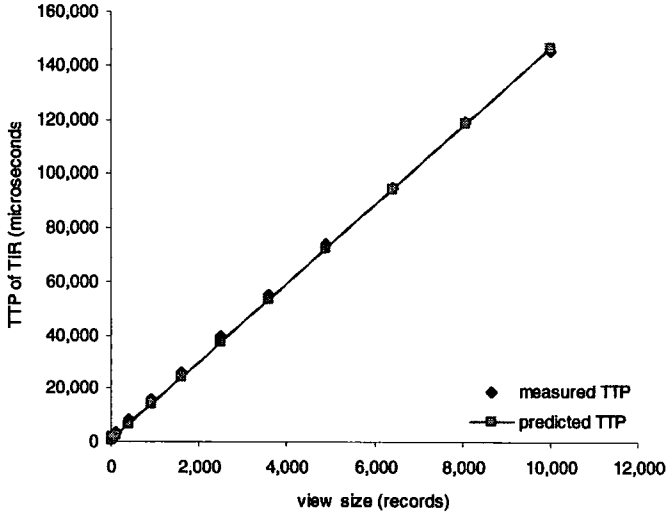


Figure 5-8 Measure and Predicted TTPs of pATIR

Table 5-10 Reconstruction and Verification (ATIR): parameters

Parameter	Value	Description
TRV_f	20.753 µs	A fixed overhead of reconstructing any number of elements
TRV_a	0.34111 µs/byte	An additional cost per record of reconstructing an byte

Reconstruction

To verify the model derived for reconstructing results (i.e. TRV), we use a linear regression model on the data to determine the regression parameters which are shown in Table 5-10. Figure 5-7 shows the time taken to reconstruct and verify the results against the increased (optimised) result sizes. The predicted values match well with the measurements. The co-efficient of determination is 98%.

5.3.2 Varying View Sizes

Figure 5-8 shows the measured and predicted TTPs of pATIR against view sizes. The predicted values are calculated following the performance models presented in Section 5.1. The values of each component are calculated through the use of primitive performance model parameters presented in Section 5.2.3 and the ATIR component model parameters presented in Section 5.3.1.

The view size used in an ATIR service is an indication of the level of privacy protection achieved. In the service, a user only needs one record in the view. As the view size grows, the protection of the user's privacy increases. That is because the view size directly reflects the number of records involved in the server side computation. As the number of records involved in an ATIR computation grows, the server has less information about the actual record that the user is interested in.

The TTPs clearly increase linearly as the view sizes increase. The figure shows that there is little difference between the prediction and the measurements of TTPs of ATIR. The co-efficient of determination is 99.96%. As the prediction for the TTPs of APIR, the relative prediction error of pATIR quickly converges to the X aixs as the view sizes increase, which is shown in Figure 5-9. That indicates that the accuracy of the prediction of TTPs of ATIR increases as the size of a view involved in the server side computation increases.

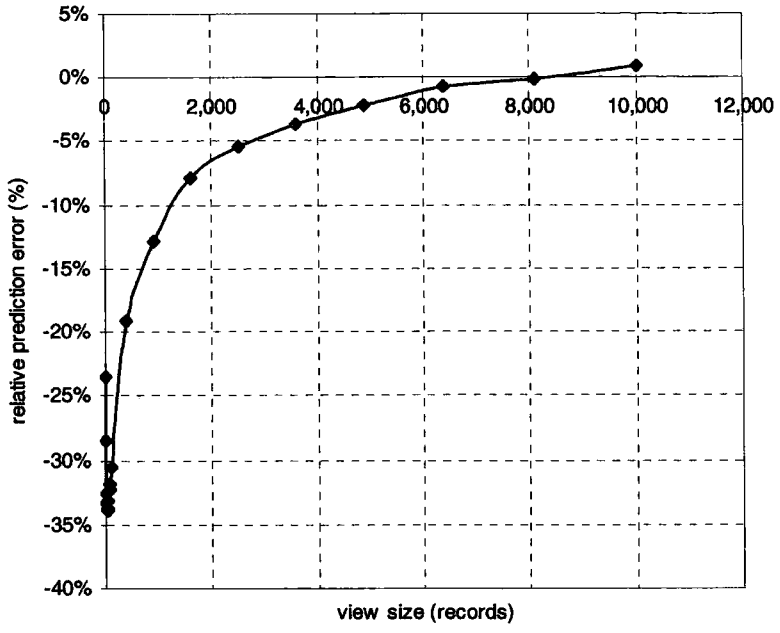


Figure 5-9 Relative Prediction Error of the TTPs of pATIR

5.3.3 Varying Result Sizes

This section investigates the impact of varying result sizes on the TTPs of pATIR. In the previous experiments, we fix the result size to be 10 bytes. With the use of optimisation techniques, the actual result size reduces 6 bytes (because the spaces at the end of the fields are trimmed). Figure 5-10 shows the impact of increased result sizes on the TTP of pATIR for varied result sizes and view sizes. In total, the figure plots ten different view sizes along with varied result sizes.

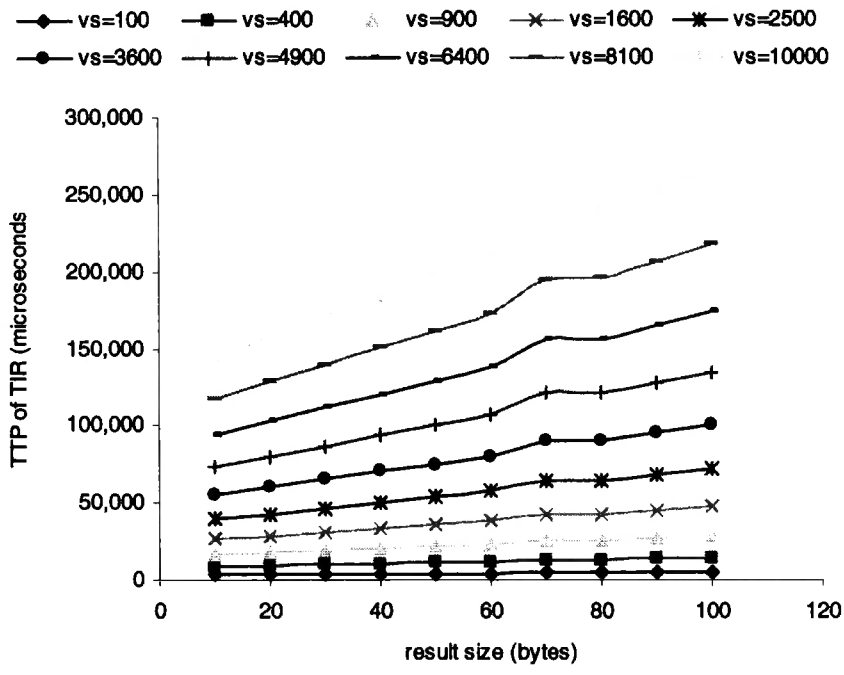


Figure 5-10 Measured TTPs of pATIR vs. Original Result Sizes

As the result sizes increases, so does the amount of data set involved in the server side computation. The trend shown in the figure clearly confirms the results obtained in Section 5.3.2 that TTPs grow linearly as the amount of data set involved in the server side computation increases. However, for small view sizes, such increment is not as significant as that of large view sizes.

When the result size hits 70 bytes, the TTPs of all view sizes exhibits a sudden increase (ranging from 1.2% of view size of 100 records to 4.7% of view size of 10,000 records) but returns to a linear trend when it reaches the result size of 80 bytes. It is independent of the actual sizes of data sets and clearly only relates to this particular result size. After tracking down the proportion of the time taken by each component of the system, we find that the symptom is caused by server side computation rather than client side computation. In particular, the sudden increase of TPA (the time taken to prepare answers) causes the sudden increase of TTPs of the result size of 70 bytes. From the process of tracking down the problem, we believe that there are two possible sources which may cause this problem. First, it may be due to an implementation bug of the ATIR system. Second, it may also be a problem caused by the underlying operating system. However, the exact cause is still under investigation.

5.3.4 Varying Undetected Error Rates

The order of finite fields, i.e., p, in pATIR determines the fault tolerance capability of an ATIR scheme and is inversely proportional to the *Undetected Error Rate* (UER). UER is specified by a user and is the level of errors that the user wishes to tolerate. In the current implementation, UER is specified at a *character* level rather than a result level. For example, if we retrieve one result which contains one character and the UER is 0.30, there is 30% chance that the reconstructed result (in this case, one character) is valid but not correct.

When a result contains more characters, the overall UER is the multiplication of the UER of each tampered character in the result. For example, if four characters in a result is tampered and the UER for each character is 30%, the UER for the reconstructed result is $0.0081 = (30\%)^4$, which means that the reconstructed result has only 0.81% chance to be a valid but incorrect one. In terms of fault tolerance, this implies that the more characters of a result being tampered, the easier for the system to detect the corrupted result. This is, however, a theoretical explanation.

Table 5-11 shows the relationship between UER and the size of the finite prime fields that are used in the experiments of this section. The calculation is based on the following settings: the valid range of reconstructed characters is the integers in the interval of [0, 255] and the number of replicas is three, which tolerates one faulty server. This value of k, we believe, is expected to be sufficient for most real applications because, with more replicas, the configurations and resource consumptions may become excess for real world applications.

Each element of a finite field in our ATIR implementation is represented by a two-byte unsigned integer, whose range is from 0 to 65,535. This range is sufficient to cover the most stringent fault tolerance requirement that our system designs for. For example, when the UER is 0.01 (that is, there is only 1% chance that a reconstructed character is valid but not correct), the corresponding size of a finite field is 25523.

Figure 5-11 illustrates the trend of TTPs of ATIR with increased view sizes with varying undetected error rates. For a given view size, the figure clearly suggests that the variance of UER has little impact on the TTPs of ATIR. The implication of this outcome is significant because it suggests that we can have a higher-level fault tolerance without compromising performance.

Table 5-11 The Mapping between UER and *p*

UER	0.01	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
P	25523	2557	1277	853	641	521	431	367	331	293	257

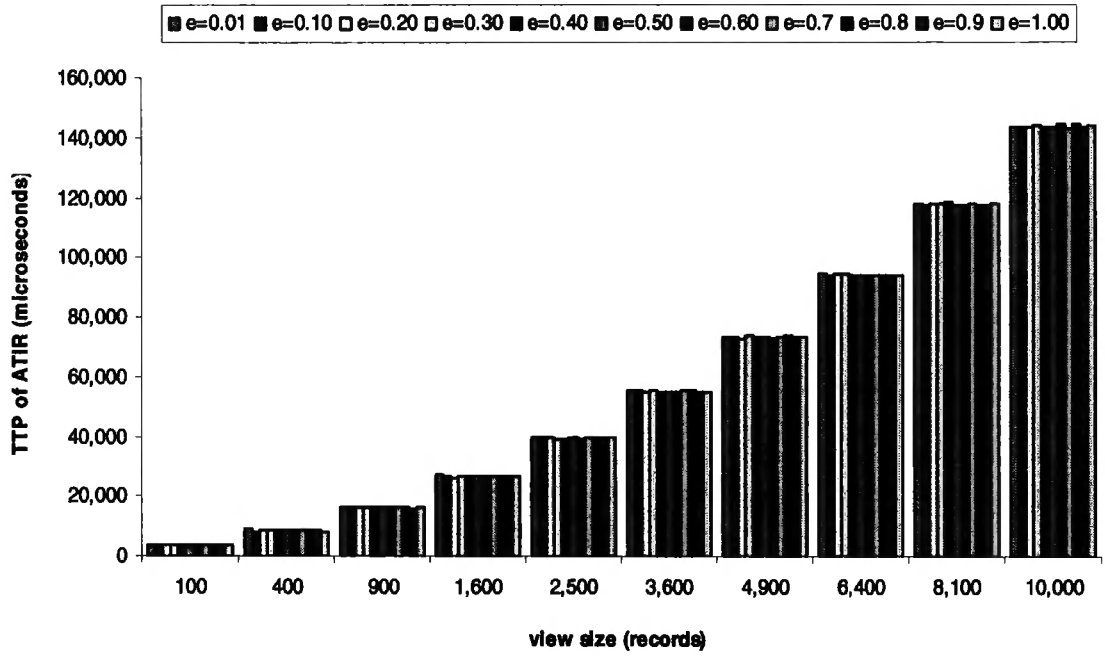


Figure 5-11 TTPs of pATIR with Varying Undetected Error Rates

It should be pointed out that the variance of UER only change the time taken to perform certain ATIR operations (e.g. query preparation and answer calculation) while the time taken to do other operations, such as TCV, remain the same. From an operating system's point of view, a finite field operation involves two unsigned 16-bit integers, which is true despite the actual values of these integers. Furthermore, the number of operations does not change as a result of varied UER. Therefore, the size of a finite field has no influence on the time taken to do the operation.

Although the figure (5-11) suggest there is not much difference between the TTP of the system with different UERs, the variance of UER does makes an impact on certain components of TTP. A closer examination of the component values reveals that the TRV (i.e., reconstruction and verification) cost is inversely proportional to UER. The smaller UER is the longer TRV takes. It is because with the finite fields use prime numbers, rather than 257, as its order (i.e., size), the time taken to perform field operations (in particular, reverse operations) is higher than using the field

GF(257). This is an implementation specific issue because we store the corresponding reverse element in the memory for each element in the field GF(257). However, with a larger prime field, the system has to perform the reverse calculation on the fly and the time taken to perform such computation is inversely proportional to the size of the field. The bigger the size the longer it takes. Note that bigger finite field sizes correspond to smaller UER. But because TCV is a small portion of TTP and due to the scale of the figure, such difference is masked by the figure.

5.3.5 Performance Proportion of Each Component

To identify the major sources of performance overheads, this section describes how each component contribute to the overall TTPs of ATIR for fixed result size 10 bytes. Figure 5-12 shows the proportion of each component of ATIR in the TTP as the view sizes increase. In particular, the proportion of the time taken for server side computation (i.e., TPV and TPA) quickly becomes the dominator factor of the TTPs. As the view sizes increase, it is also clear that the proportion of communication time rapidly reduces from 23.27% (view size = 1 record and data size = 0.01 KB) to 1.40% (view size = 1,600 records and data size = 16 KB).

The figure also shows us that TRV vanishes quickly as the view size increases. When the view size is one record, TRV takes up 1.97% of the TTP time. When the view size becomes 10,000 records, the proportion drops to 0.02%. Although the data in the figure uses a fixed result size, it does give us a clear implication of how TRV compares to TTP as the view sizes increase.

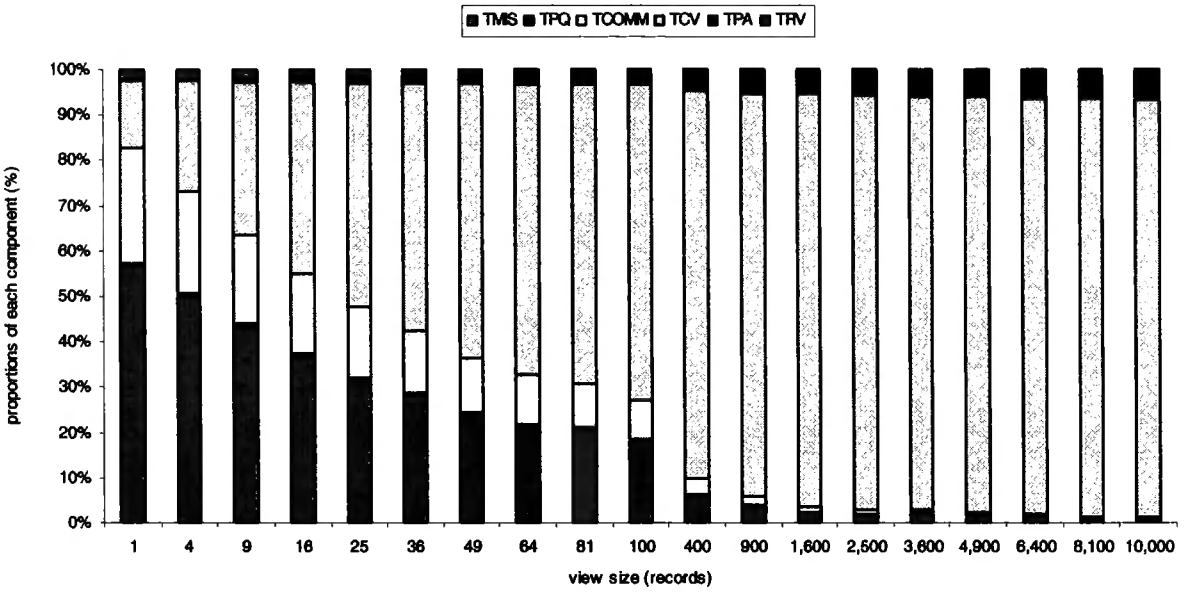


Figure 5-12 Performance Proportion of Each Component in TTP of ATIR

5.3.6 Performance of Deterministic ATIR

So far, we have been focused on the probabilistic ATIR. This section compares the TTPs of a deterministic ATIR (dATIR) and a probabilistic ATIR (pATIR). dATIR ensures that the delivered result is 100% to be the correct one whereas pATIR only offers a probabilistic guarantee of the safety property. Both ATIR implementations differ in the way that the reconstruction and verification algorithm is implemented. pATIR stops and returns a result once it finds a valid result whereas dATIR reconstructs all the possibilities of results and determines whether there is a correct result. When distinct valid results are more than one, dATIR aborts and reports a failure. Since the system can reconstruct at least one correct result, if only one result remains, it must be the correct one. In terms of number of reconstruction attempts needed, dATIR represents the worse case scenario of pATIR. So, clearly, the TRV of dATIR is longer than that of pATIR. But as we have shown in the previous section the proportion of TRV in TTP of pATIR is small.

We conduct ten series of independent dATIR experiments as follows. The view sizes in the series increment from 100 to 10,000 records. In each series, we vary the result sizes from 10 to 100 bytes in each test. Each test is repeated eleven times and the first result is ignored. Each value reported is the average of ten tests. At the end, we have 100 measurements for the ten series of experiments.

Figure 5-13 compares the pATIR with the dATIR when data sizes involved in the server side computation are small, that is, the data sizes range from 1,000 bytes to 10,000 bytes. Figure 5-14 does the same comparison with large data sizes (from 100KB to 1MB). Overall, the dATIR adds some performance overhead to the TTP time. Over 96% of the observed dATIR TTP measurements are larger than that of pATIR, which suggests that dATIR does have some performance overhead, comparing with pATIR. The increments are, however, not the same for all experiments. As shown in these figures, the increments of dATIR are much significant (as high as 13.43% for 1,000 bytes data) for small data sizes than those for large data sizes (consistently lower than 0.8% when data sizes are 1K bytes to 1M bytes). This is mainly because of the scale of the measurements. For small data sizes, the increments are compared with small quantities (several milliseconds) of TTPs. For large data sizes, the increments are compared with large quantities (several hundreds of milliseconds) of TTPs.

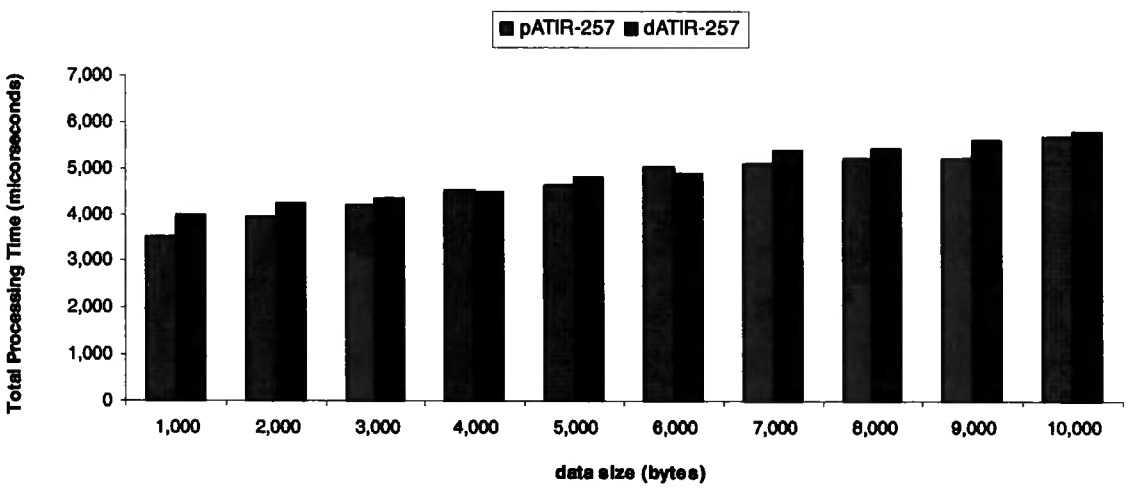


Figure 5-13 dATIR vs. pATIR (Small Data Sizes)

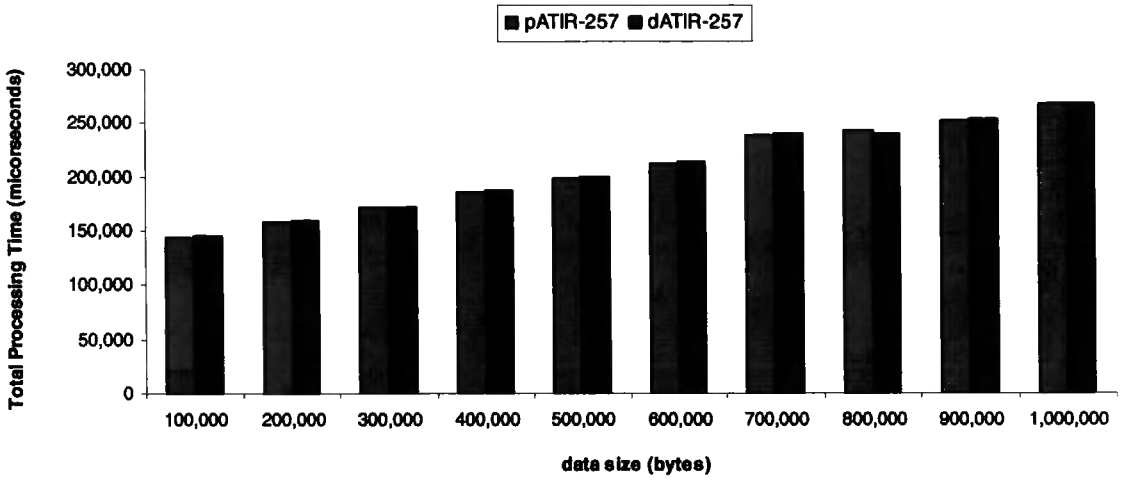


Figure 5-14 dATIR vs pATIR (Large Data Sizes)

5.4 ATIR Experiments in a Simulated Faulty Environment

This section describes ATIR performance in the presence of the following simulated faults: crash faults and malicious faults.

5.4.1 ATIR Performance in the Presence of Crash Faults

In this experiment, one of the three servers is shut down and only two servers provide ATIR services. Figure 5-15 compares the TTPs of ATIR in normal and crash failure situations when the result size is set to be 10 bytes. It is clear that the occurrence of crash faults has little impact on the measured TTPs of the ATIR system. In some cases, such as when the view size is 8,100 records, the observed TTP in the presence of crash failures is even lower than that of ATIR in normal situations. It is because the system adapts to the situation and uses the answers from the available servers to reconstruct the result. The server fault was detected by the socket connection operation of the client program and therefore the result from this particular server will not be waited for. Since the machines are identical and any two servers are sufficient to reconstruct a result, the unavailability of one server does not have much influence on the TTP of the system. We expect this situation remains for other result sizes.

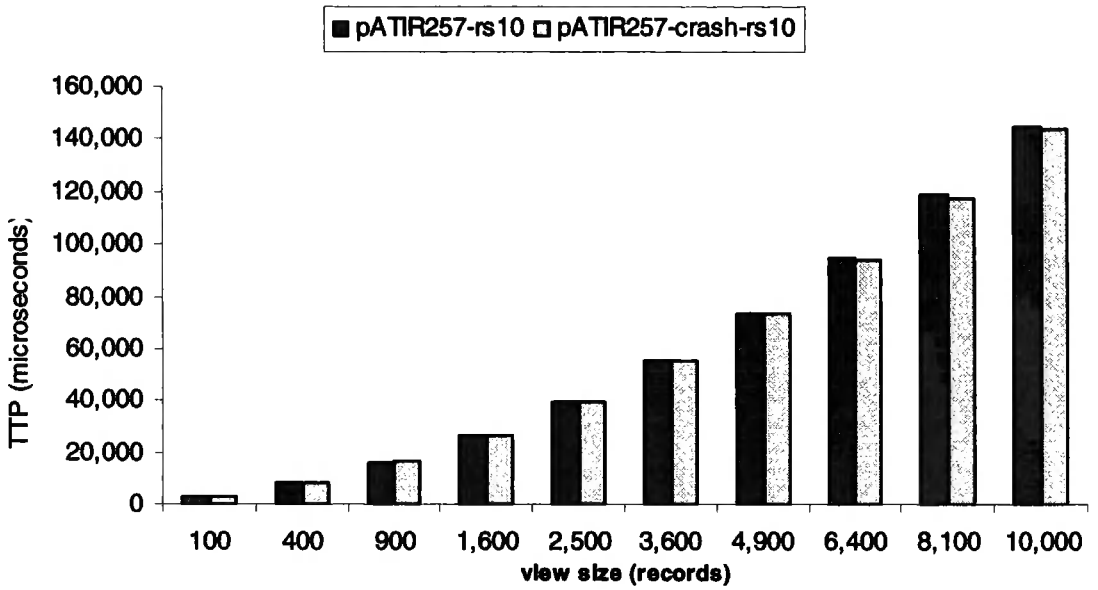


Figure 5-15 ATIR in Normal vs. Crashed Situations

5.4.2 ATIR Performance in the Presence of Malicious Faults

Having shown that the system maintains good performance in the presence of crash failures, we now investigate the performance and behaviours of the system when malicious faults occur. As a security system, what we really concern with is how an attacker can effectively attack the system. Any system is built on assumptions and therefore, invalidating assumptions is one of the effective ways of compromising the system. However, such information is often not readily available. For a system that implements a new security scheme, such as ours, it is important to ensure that we know the weakness of the system.

In fact, the behaviours of malicious attackers are hard to determine and difficult to predict. Therefore, the most important thing we need to make sure when it comes to simulating malicious faults and experimenting with the system in such situations is to ensure that the simulation is fair. That is, we are not choosing an attack strategy that is in our favour. In other words, we shall identify the types of attacks that our system is susceptible to. Once we have this information, the attack simulation should be based on such attacks to experiment with the system. That is effectively how it works in real world. Once an attacker finds out an effective attack strategy that can be taken advantage of, it is certain that the new strategy will be used rather than relying on any exiting ones.

Simulating Malicious Failures

Our system is most susceptible to malicious value faults, in which one or more compromised servers deliver purposefully manipulated answer(s). We simulate this type of attacks by randomly flipping the characters in an answer right before it is returned to the client.

The simulation has two purposes. First, we need to find out the optimal value for achieving a required level of security through experiments. To our system, that means how to set the UER parameter for given a user security requirement. Second, since the system performance in normal situations is already known, it is mandatory to find out whether the performance will degrade as a result of such attacks.

In the experiments, we do not single out the time taken to do the simulation since it is negligible. It only adds several microseconds to the TTP of the system. To simulate the attacks, we add no more than 10 lines of code into the client program and add one line of code into the server program. Hence, the impact of injecting the faults can be ignored.

To avoid over simplifying the possible attacker behaviours, we need to answer two further questions as follows:

- What is the best strategy for an attacker to attack the ATIR system? As indicated in Section 3.3.4, we know that the more characters being changed the easier for the client to detect the corrupted result. Is this true for the real experiments?
- To what extent, the TTPs of ATIR will degrade in the presence of malicious attacks.

Finding out the Actual Undetected Error Rates

To address the first question, five series of experiments, with different number (i.e., one to five) of characters being *randomly* flipped just before an answer is returned, were conducted. Each series of experiment contains 19 tests in which the undetected error rate starts from 0.01, 0.02 to 0.10 and then 0.20 to 1.00. In each test, we varied the view sizes from 100 to 10,000 records. Each test is repeated eleven times and the first result is ignored. In each test, we log down the following information: TTP and UER.

Since there is only one out of three servers that involves in the simulated attacks, the client will have sufficient correct answers to reconstruct a valid and correct result. The most dangerous situation is that the client delivers valid but incorrect results. Therefore, we search through the log and identify the maximum undetected error rate with which no valid but incorrect results are observed.

Figure 5-16 shows the relationship between the required (maximum) undetected error rates and the number of characters that are randomly flipped on the server side. The picture tells us that the required UER increases as the number of flipped characters increases. That means when an attacker modifies more characters, a large UER is good enough to detect invalid results and identify the correct ones. In other words, it is easier for the client program to detect the occurrence of attacks when more characters are manipulated even with large UERs. Therefore, this provides an affirmative answer to our first question, that is, the best strategy an attacker should follow is to modify as few characters as possible. Otherwise, the client can easily detect the occurrences of attacks even with large UERs.

On the other hand, this figure also reveals that when the number of flipped characters exceeds a certain level, in this case, four characters, a same level of undetected error rates are sufficient to detect all the errors. This provides an affirmative answer to our first question arise in the last section. That is, the more characters an attacker tampers, the easier the ATIR system detects the occurrence of tampering. Therefore, in order to conceal the act of attacking the system, the best strategy an attacker should take is to tamper with fewer characters.

TTPs in the Presence of Malicious Attacks

To answer the second question, we need to go back to the fundamentals of our scheme. We now describe how much extra time is required for the system to deal with the value attacks. Figure 5-17 compares the TTPs of the system in normal situations and in the presence of (simulated) malicious attacks with varied view sizes. Six groups of experiments with a different number of flipped characters are performed and each column in the figure represents the results obtained for each group of experiment. For example, "e=0.01-normal" means that no character is flipped and they are the experiments conducted in a normal situation. "e=0.01-1char" means that one character is randomly flipped to simulate value attacks on the system.

In all experiments, the undetected error rate is set to be 0.01. The figure suggests that there is little difference between the TTPs in the different situations. Effectively, that means that the occurrence of malicious failures has little impact on the system performance.

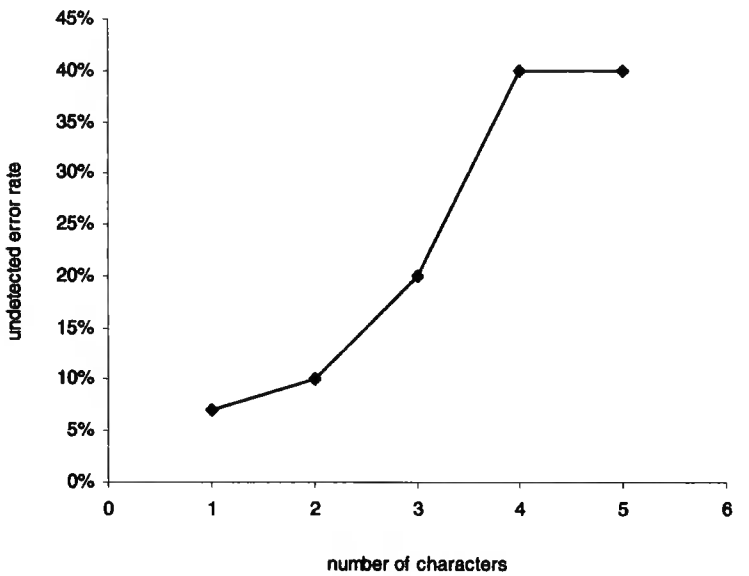


Figure 5-16 Undetected Error Rate vs. Number of Flipped Characters

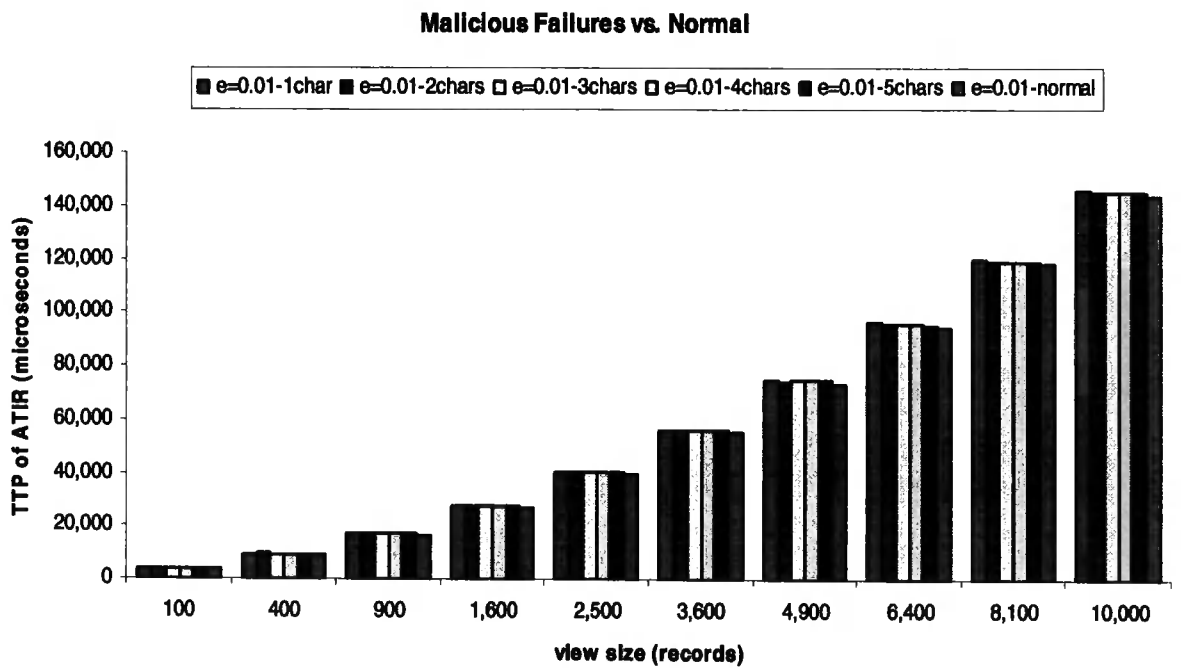


Figure 5-17 TTP of pATIR in Normal and Malicious Faults

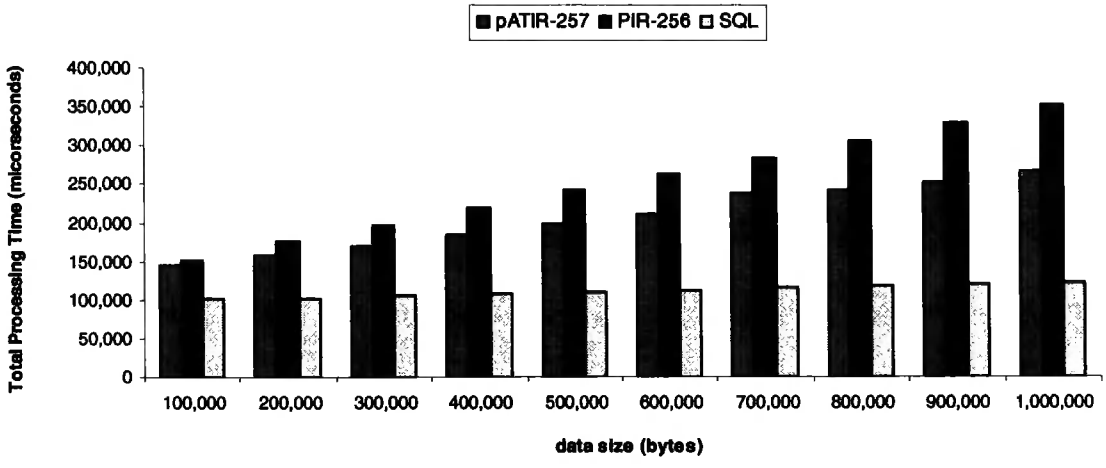


Figure 5-18 TTPs of PIR-256, pATIR-257, and SQL

5.5 Performance Comparisons among SQL Query, PIR and ATIR

In this section, we compare the TTPs of using three query services: SQL, PIR, and ATIR to retrieve data from remote server (s). A normal SQL query with varied view sizes is executed between two separated computers in the same network environment to download the given amount of data. Therefore, in this experiment, the use of SQL queries is equivalent to the use of downloading to provide privacy protection.

Figure 5-18 compares the TTPs of using ATIR, PIR and SQL technologies to query remote databases. The sizes of data sets involved in the queries range from 100KB to 1 MB.

Firstly, the TTPs of all services increase linearly as the data size increases. The increment rates, however, are significantly different. Compared with querying 100KB data, it is observed that it costs 85%, 132%, and 20% more for ATIR, PIR, and SQL services to query 1MB data, respectively.

Secondly, ATIR consistently shows a better performance than that of PIR in all these experiments. As the data size grows, the superiority becomes even obvious. For example, when the data size is 1MB, it takes the ATIR implementation 24% less TTP time to complete the operation comparing with using the PIR implementation. It is clearly not a coincident event that can be caused by (random) system errors because of the consistency shown in the figure. It is due to the different computational operations required by PIR and ATIR schemes and the

implementations of finite field operations, which subsequently lead to different implementation strategies used in the systems. As mentioned in Section 4.2.1, PIR uses a finite power field GF(256) for computation whereas ATIR uses a finite prime field GF(257). Putting it simply, the calculation operations (in particular, reverse operations) in GF(257) takes less steps than those in GF(256). This is the major reason for the better performance of ATIR over PIR in all these experiments.

Finally, the observed TTPs of both PIR and ATIR are all well under 0.5 second for data sets of sizes ranging from 100KB to 1 MB. These results make both technologies appealing for real applications.

5.6 Summary

This chapter first derives performance models to model the major components of an ATIR system. These models are then used to predict the TTP of ATIR. We show that the component prediction models are accurate since the predicated results match well with the measurements. Most predictions achieve a high co-efficient of determination, often above 97%. We also show that the communication overhead plays a negligible part of the overall TTPs in ATIR.

We then present significant experimental studies to evaluate the performance of the implemented ATIR system in normal and simulated faulty environments. In particular, the experiments examine the impacts of varying the parameters of ATIR on the system performance. We further study the performance contributions of each major component of an ATIR system and compare the performance between pATIR and dATIR.

The effectiveness of attack tolerance capability of ATIR schemes is evaluated by simulating various attacks on servers. The major experimental results reveal the following: i) crash faults make little impact on the performance of an ATIR system; and ii) the system performance maintains at the same level even in the presence of simulated malicious attacks.

Finally, we compare the performance among SQL, PIR and ATIR and observe that the TTP of PIR increases most significantly as the sizes of data sets grow. We also show that the TTPs of both PIR and ATIR are well under half a second for performing the operations over data sets of sizes up to 1 MB.

Chapter 6 Conclusions and Future Work

Our fast-growing reliance on online query services demands an appropriate level of privacy protection as well as highly available service provision. These problems are often treated separately. The prevalence of malicious attacks on online services calls for a balanced solution to satisfy both requirements. This thesis has developed new ATIR schemes for performing certain database queries privately and correctly despite the occurrence of attacks, has addressed the problem of privacy protection as well as service provision for certain types of database query applications, has explored the practical ways for designing and implementing ATIR systems, and has presented significant evaluation results for demonstrating the effectiveness and practicability of the implemented ATIR systems.

This chapter is organised as follows. In Section 6.1, we summarize the main contributions made by this thesis to the following three areas of research: PIR, security, and fault tolerance. At the same time, we also revisit the research challenges set out in Chapter One and spell out the results we have achieved in this thesis on tackling these challenges. The theoretical and practical results of ATIR spark a number of interesting research questions for future research, which are outlined and discussed in Section 6.2. Finally, Section 6.3 concludes the thesis.

6.1 Main Contributions

This thesis has demonstrated a systematic and sound approach for constructing ATIR schemes, implementing ATIR systems, and evaluating ATIR systems through extensive experiments. In this thesis, two advanced ATIR schemes are developed for performing certain database queries privately and correctly even in the presence of certain attacks. Compared with existing PIR schemes, ATIR works in a much more realistic setting by taking malicious attacks into account while maintaining a balanced level of communication efficiency. By enabling privacy protection and simultaneously achieving a high level of service provision, ATIR represents a new

approach for constructing secure and fault tolerant schemes and designing privacy-preserving and highly available systems. In order to investigate the practicability of the schemes, ATIR has been evaluated extensively through empirical studies in various conditions. Along with a revisit on the research challenges set out in Chapter One, we summarise the major results obtained by this thesis as follows:

- A survey of the major techniques and the state of the practice systems of coping with attacks has been given to set the context of this thesis. In order to characterise the power of different types of attackers, two attack models are derived and their relationship is examined (Section 2.2). The characterisation is followed by a critical review of privacy protection techniques, PIR research, and attack tolerant techniques and systems (Section 2.3, 2.4, 2.5). In particular, some commonly used assumptions on design and implementation of attack tolerant systems are critically reviewed and examined (Section 2.5.5).

- ATIR is novel because it offers privacy protection for users as well as ensuring service availability even in the presence of malicious attacks. In particular, ATIR can tolerate *any collusion* of up to t servers for privacy violation and up to f faulty (crashed or malicious) servers in a system with k replicated servers, provided that $k \geq t + f + 1$ where $t \geq 1$ and $f \leq t$. Albeit a specific type of privacy protection, none of existing fault tolerant systems provide such protection for users against colluded servers. (Section 3.3.2, 3.3.4, 3.3.5, 3.3.6)

- In contrast to other related approaches, ATIR relies on neither enforced trust assumptions, such as the use of tamper-resistant hardware and trusted third parties, nor an increased number of replicated servers. While the best solution known so far requires k ($\geq 3t + 1$) replicated servers to cope with t malicious servers and any collusion of up to t servers, with an $O(n^{\frac{1}{\lfloor (t-1)/3 \rfloor}})$ communication complexity, ATIR uses fewer servers with a much improved communication cost, $O(n^{\frac{1}{2}})$ (where n is the size of a database managed by a server). (Section 3.3.7, 3.4) In particular, this contribution answers three research challenges - **Communication Complexity, Trust Assumptions**, and **Use of Replication** - set out in Chapter One.

- This thesis provides the details of integrating and implementing ATIR on realistic database systems. Specifically, we describe the ATIR system architecture and

show how to realise the character database model of ATIR on realistic databases for ATIR computations (Section 4.2.1). This paves the way for practically implementing ATIR as a real service. Since PIR can be viewed as a special case of ATIR in that there is no faulty server in the system. The ATIR implementation described in this thesis also demonstrates a purely software based approach for implementing PIR systems on realistic databases (Chapter 4). In particular, this contribution answers the research challenge – **Implementation** - set out in Chapter One.

- Although the processing costs of PIR schemes is envisaged to be exceedingly high, this thesis, for the first time, shows that no pre-processing or shuffling is needed for ATIR systems and the experimental results reveal that ATIR performs well. In a LAN environment, it takes well under half a second to use an ATIR service for the calculations over data sets of size up to 1MB (Section 5.5). The performance of the ATIR systems remains at the same level, even with the occurrence of server crashes and malicious attacks. (Section 5.4) In particular, this contribution answers the research challenge – **Processing Costs** - set out in Chapter One.

6.2 Future Research Directions

6.2.1 Reducing Computation Complexity

In order to protect the user's privacy, ATIR involves a large quantity of computation (e.g. in the order of n). In practice, it may be difficult to find a service provider who is willingly to allocate such a level of computation resources for the privacy and result availability of *users*. Hence, an interesting and important future work is to reduce the processing costs of servers and provide a negotiation mechanism between a client and service providers to choose the level of resources they wish to assign to perform ATIR operations.

Instead of purely hardware-based or purely software implementations, a combination of secure hardware and software processing may reduce the overall computation costs. Only security-sensitive operations are performed inside the piece of secure hardware, just like some existing secure and fault tolerant systems BFT [Cas01], COCA [Zhou01]. The installation of multiple secure co-processors may

also achieve load balancing and consequently improve overall performance of an ATIR system.

6.2.2 Reducing Communication Complexity

Currently, the communication complexity of ATIR schemes is independent of the number of replicas used. The construction of existing PIR schemes show that this problem may be resolvable with the use of recursive techniques. For example, several PIR schemes (e.g. [Amb97] and [BIKR02]) apply recursion to achieve lower communication complexity in a replication setting.

In [IK99], Ishai and Kushilevitz present PIR schemes with $O(k^3 n^{1/(2k-1)})$ communication complexity using replication-based secret sharing techniques [ISN87]. It remains unclear whether the same technique can be applied to our ATIR schemes to achieve lower communication complexity.

At the time this thesis is written, despite the improvement on the asymptotical communication complexity of PIR schemes, the communication complexity of 2-server PIR schemes remains the same, i.e. $O(n^{1/3})$ as the result obtained by the seminal paper [CGKS95]. Several techniques (e.g. covering codes in [CGKS95], recursion in [Amb97], and emulation techniques in [IK99]) are used to achieve that communication complexity. It will be interesting to explore these techniques for further reducing the communication complexity of both PIR and ATIR schemes.

6.2.3 The Importance of Design Diversity and Assumptions

The bounded numbers of curious and faulty servers are of little meaning if common-mode failures [AK84] occur in the ATIR system, because an attacker can easily exploit a common vulnerability in all servers to compromise the entire system. Traditionally, design diversity [AK84], data diversity [AK88, TMB80], and environment diversity approaches [HK93] have often been considered to cope with such failures.

It would be helpful to reduce the overall vulnerability of the system by incorporating these design diversity approaches into the system design and implementation. For example, in our systems, diverse operating systems (Linux and Windows) and programming languages (e.g. Java and C) can be used for the implementation. We can also choose from a wide range of readily available commercial database engines (e.g. MySQL and Microsoft SQL Server) to integrate

with servers. In fact, apart from the C implementation, ATIR systems have a demonstration version which is implemented in Java. Both implementations conform to the same interface and design but with different implementation strategies (the C implementation integrates with MySQL C APIs whereas the Java implementation uses the JDBC Driver for MySQL.).

6.2.4 Exploring Other Applications

We have shown that the ATIR system performs well for certain database queries in normal as well as simulated faulty cases. The next task for this research is to find a wider range of applications that can take advantage of this technology. It would be appealing to adapt and incorporate the existing implementations with other applications, such as certification authority, public data repository, and patent query services. Among them, certification authorities are the most interesting because they offer an infrastructural security service underpinning most distributed systems. Due to the increasing popularity of large scale distributed systems, the use of certificates has become popular. Hence, privacy-preserving, secure and fault-tolerant online certification authorities will be of real interest in practice. The Cornell Online Certification Authority (COCA) [ZSR02] is a latest effort along this direction. But COCA doesn't provide any privacy protection for users in the presence of malicious attacks. Once a server is compromised, all the messages exchanged between the user and the servers are exposed to an attacker.

A further question is to investigate whether there are any other domains that ATIR can be used. One of the fundamental difficulties of adapting ATIR to other appropriate domains is the restricted semantics of the ATIR operations, as discussed in Section 4.1 and 4.3.1.

6.2.5 Defending against Denial-Of-Service Attacks

To some extent, ATIR has an inherent capability to deal with Denial-Of-Service (DOS) attacks. Servers that are compromised by DOS attacks respond slower than the normal servers do. Due to the adaptive nature of our scheme, the client program in ATIR simply ignores the slow servers and use the answers from the good servers to reconstruct results, assuming good servers perform as good as they do in normal situations. However, it is important to incorporate design diversity into various stages

of system design, as discussed in Section 6.2.3. Otherwise, attackers can easily exploit the common vulnerabilities of an ATIR system and hack into the system.

6.3 In Conclusion

In this thesis, we have successfully demonstrated a systematic approach for investigating and studying the feasibility, usefulness, and practicability of ATIR – a privacy-preserving and fault tolerant mechanism for secure information retrieval. The soundness of the research conducted by this thesis is supported by the following: i) a significant literature survey on the state-of-the-art techniques and systems on privacy protection, PIR, and attack tolerance; ii) a detailed presentation of several ATIR schemes with complete characterisations of fault tolerance conditions and privacy protection properties; iii) analytical comparison of ATIR with relevant PIR efforts and critical examinations amongst ATIR and state-of-the-art secure and fault tolerant systems; and iv) an extensive experimental evaluation of ATIR systems in both normal and simulated faulty environments, with good performance results.

As a whole, we have shown that ATIR offers an attractive and practical solution for ever-increasing online information applications. We believe that many principles of ATIR may be extended to a wider range of application domains. As with any other research disciplines, there still remains much research to be done on deploying ATIR in real world applications.

Glossary

Attack An attack is an intentional fault aiming to violate the security properties of a service. There are two kinds of attacks: passive attacks and active attacks. Passive attacks seek to disclose confidential information whereas active attacks not only disclose confidential information but also disrupt services that are provided by a computer system.

Attack Model An attack model is an abstraction of all the types of attacks that are considered by a computer system.

Attacker An attacker is a person or a computer program who carries out attacks against a computer system.

Component is an entity of a system.

Error An error is the part of a system state that is liable to lead to subsequent failures.

Failure A failure is the manifestation of an error of a system. A system failure occurs when the delivered service deviates from what a system is aimed at (e.g., specification).

Fault A fault is the hypothesised cause of an error.

Result A result is the intended data item (s) that a user wants to retrieve from a database. For example, a result can be the parts of a record or an entire record stored in a database.

View A view is a collective representation of all the records involved in the server side computation in an ATIR/PIR system.

Reference

[AF02] D. Asonov and J-C. Freytag, "*Almost Optimal Private Information Retrieval*", in Proc. 2nd Workshop on Privacy Enhancing Technologies (PET2002), San Francisco, USA, April 2002.

[AFK87] M. Abadi, J. Feigenbaum, J. Kilian, "*On Hiding Information from an Oracle*", in Proc. 19th Annual ACM Conference on Theory of Computing, 1987, pp. 195-203.

[AK84] A. Avizienis and J. P. J. Kelly, "*Fault Tolerance by Design Diversity: Concepts and Experiments*", IEEE Computer, 17(8): 67-80, August 1984.

[AK88] P. E. Amman and J. C. Knight, "*Data Diversity: An Approach to Software Fault Tolerance*", IEEE Transactions on Computers, 37(4): 418-425, 1988.

[ALN87] N. Ahituv, Y. Lapid, and S. Neumann, "*Processing Encrypted Data*", Communications of the ACM, 30(9), pp. 777-780, 1987.

[Amb97] A. Ambainis, "*Upper Bound on the Communication Complexity of Private Information Retrieval*", Proc. 24th International Colloquium on Automata, Languages and Programming (ICALP'97), LNCS, vol. 1256, Springer-Verlag, Bologna, Italy, July 1997, pp. 401-407.

[And01] R. J. Anderson. "*Security Engineering: a Guide to Building Dependable Distributed Systems*", New York ; Chichester : John Wiley, c2001.

[Ano04] Anonymizer, an online tool for providing private web surfing service, www.anonymizer.com.

[Appel02] A P3P Preference Exchange Language 1.0 (Appel 1.0), working draft, W3C, Apr. 2002.

[Aso04] D. Asonov, "*Querying Databases Privately – a New Approach to Private Information Retrieval*", LNCS 3128, Springer, 2004, ISBN: 3540224416.

[Bei04] A. Beimel, Personal Communication via emails, 2004.

[BF90] D. Beaver and J. Feigenbaum, "*Hiding Instances in Multioracle Queries*", in Proc. 7th Annual Symposium on Theoretical Aspects of Computer Science (STACS'90), Rouen, France, LNCS, vol. 415, Springer-Verlag, Feb.1990, pp. 37- 48.

[BFK00] O. Berthold, H. Federrath, and S. Kopsell, "*Web MIXes: A System for Anonymous and Unobservable Internet Access*", in Designing Privacy Enhancing Technologies, LNCS Vol 2009, pp. 115-129, Springer-Verlag, 2000.

[BI01] A. Beimel and Y. Ishai, "*Information-Theoretic Private Information Retrieval: A Unified Construction*", in Extended Abstract in ICALP 2001, vol. 2076 of LNCS, pp. 89-98.

[BIKR02] A. Beimel, Y. Ishai, E. Kushilvitz, and J.-F. Raymond, "*Breaking the $O(n^{1/(2k-1)})$ Barrier for Information-Theoretic Private Information Retrieval*", in Proc. of the 43rd IEEE Symposium on Foundations of Computer Science (FOCS), 2002.

[BIM00] A. Beimel, Y. Ishai, and T. Malkin, "*Reducing the Servers Computation in Private Information Retrieval: PIR with Preprocessing*", in Proc CRYPTO 2000, LNCS vol. 1880, pages 56-74, Springer-Verlag, 2000.

[BJN00] D. Boneh, A. Joux, and P. Nguyen, "*Why Textbook ElGamal and RSA Encryption are Insure*", in Proc. AsiaCrypt'00, LNCS vol. 1976, pp. 30 – 40, Springer-Verlag, 2000, available at: http://crypto.stanford.edu/~dabo/papers/ElGamalattack.ps.

[Bla79] G. R. Blakley, "*Safeguarding Cryptographic Keys*", in Proc. National Computer Conference, American Federation of Information Processing Societies Proceedings 48, 1979.

[Bla83] R. Blahut, "*Theory and Practice of Error Control Codes*", Addison-Wesley Publishing Company, 1983, ISBN 0-201-10102-5.

[Bre01] T. Brekne, "*Encrypted Computation*", Ph.D. thesis, Department of Telematics, Norwegian University of Science and Technology, July, 2001.

[BS02] A. Beimel and Y. Stahl, "*Robust Information-Theoretic Private Information Retrieval*", in Proc. 3rd Conference on Security in Communication Networks, 2002, pp. 326-341.

[BSG00] P. Boucher, A. Shostack, and I. Goldberg, Freedom Systems 2.0 Architecture, 2000, available at: http://whitepapers.zdnet.co.uk/0,39025945,60025335p-39000494q,00.htm.

[Cac03] C. Cachin, "*An Asynchronous Protocol for Distributed Computation of RSA Inverses and its Applications*", in Proc. 22nd ACM Symposium on Principles of Distributed Computing (PODC 2003), pp. 153-162, July 2003.

[Cac04] C. Cahine, "*Distributed Trust on the Internet - Position Paper for S.O.S. Workshop 2004*", in Proc. FuDiCo II: S.O.S., Survivability: Obstacles and Solutions, 2nd Bertinoro Workshop on Future Directions in Distributed Computing, 23-25 June 2004, Bertinoro, Italy, available at: http://www.cs.utexas.edu/users/lorenzo/sos/SOS/Cachin.pdf.

[Cas01] M. Castro, "*Practical Byzantine Fault Tolerance*", tech. report MIT/LCS/TR-817 and Ph.D. dissertation, Laboratory for Computer Science, MIT, Cambridge, MA, USA, Jan. 2001.

[CDK01] G. Coulouis, J. Dollimore, and T. Kindberg, "*Distributed Systems – Concepts and Design*", Addison-Wesley Publishers Limited and Pearson Education Limited, 3rd Edition, 2001.

[CE02] S. Cantor and M. Erdos, "*Shibboleth-Architecture*", Draft v05, NSF Middleware Initiative Draft, May 2, 2002. URL: http://shibboleth.internet2.edu/draft-internet2-shibboleth-arch-v05.html.

[CG97] B. Chor and N. Gilboa, "*Computational Private Information Retrieval*", in Extended Abstract in Proc. 29th Annual ACM Symposium on Theory of Computing (STOC) , 1997, pp. 204-313.

[CGH+97] R. Canetti, R. Gennaro, A. Herzberg, and D. Naor, "*Proactive Security: Long-term Protection against Break-ins*", CryptoBytes, 3(1), spring 1997.

[CGKS95] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan, "*Private Information Retrieval*", in Proc. 36th Annual Symposium on Foundations of Computer Science (FOCS'95), Milwaukee, Wisconsin, USA, 23-25 Oct. 1995, pp. 41-51. Journal version: J. of the ACM, vol. 45, no. 6, 1998, pp. 965-981.

[CGN98] B. Chor, N. Gilboa, and M. Naor, "*Private Information Retrieval by Keywords*", Technical Report 98-03, Theory of Cryptography Library, 1998, available at: http://philby.ucsd.edu/cryptolib/1998/98-03.html.

[CGMA85] B. Chor, S. Goldwassser, S. Micali, and B. Awerbuch, "*Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults*", in Proc. 26th IEEE Symp. On Foundations of Comp. Science, pages 383 – 395, Portland, 1985.

[CH02] J. Camenisch and E. van Herreweghen, "*Design and Implementation of the IDEMIX Anonymous Credential System*", in Proc. Of ACM Conference on Computer and Communications Security, ACM, 2002.

[Cha81] D. Chaum, "*Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms*", Communications of the ACM, 24(2):-88, February 1981, available at: http://mixmaster.shinn.net/chaum-acm-1981.html.

[Cha88] D. Chaum, "*The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability*", Journal of Cryptography, 1: 65-75, 1988.

[Cha04] Y. Chang, "*Single Database Private Information Retrieval with Logarithmic Communication*", in Proc. ACISP 2004, Lecture Notes in Computer Science 3108, Springer-Verlag, pp. 50-61, 2004.

[Chi79] L. Childs, "*A Concrete Introduction to Higher Algebra*", Springer-Verlag, 1979, ISBN: 0-387-90333-x.

[CJRS89] D. D. Clark, V. Jacobson, J. Romkey, and H. Salwen, "*An Analysis of TCP Processing Overhead*", IEEE Communications Magazine, vol. 27, pp. 23-29, June 1989.

[CL99] M. Castro and B. Liskov, "*Practical Byzantine Fault Tolerance*", in Proc. Of the 3rd Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, LA, Feb. 1999.

[CLRS01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, "*Introduction to Algorithms*", 2nd Edition, MIT Press, 2001, ISBN: 0-262-03293-7.

[CMS99] C. Cachin, S. Micali, and M. Stadler, "*Computationally Private Information Retrieval with Polylogarithmic Communication*", in J. Stern, Editor, Advances in Cryptology – EUROCRYPT ' 99, Vol. 1592 of LNCS, p. 402-414, Springer, 1999.

[Cop84] D. Coppersmith, "*Evaluating Logarithms in GF*(2^n)," in Proc. 16th ACM Symposium on Theory of Computing, pp. 201 – 207, 1984.

[CP02] C. Cachin and J. A. Poritz, "*Secure Intrusion-Tolerant Replication on the Internet*", in Proc. Intl. Conf. on dependable systems and networks (DSN-2002), Washington DC, USA, June 2002.

[CS03] C. Cachin and A. Samar, "*Secure Distributed DNS*", Research Report RZ 3509, IBM Research, October 2003.

[Dan03] George Danezis, "*Mix-Networks with Restricted Routes*", in Proc. 3rd International Workshop on Privacy Enhancing Technologies (PET'03), LNCS 2760, Springer-Verlag, 2003, pp. 1-17.

[Dai04] Wei Dai, Crypto++ Library 5.2, 2004, URL: http://www.eskimo.com/~weidai/cryptlib.html.

[DBF91] Y. Deswarte, L. Blain, and J.-C. Fabre, "*Intrusion Tolerance in Distributed Computing Systems*", in Proc. of the IEEE Symposium on Research in Security and Privacy, Oakland, USA, 1991.

[DH76] W. Diffie and M.E. Hellman, "*New directions in Cryptography*", IEEE Transactions on Information Theory 22 (1976), pp. 644-654.

[DOJ00] U. S. Department of Justice, "*Identity Theft and Fraud*", 2000, available at: http://www.usdoj.gov/criminal/fraud/idtheft.html.

[FDR95] J.-C. Fabre, Y. Deswarte, and B. Randell, "*Designing Secure and Reliable Applications using Fragmentation-Redundancy-Scattering: an Object-Oriented Approach*", in Predictably Dependable Computing Systems, B. Randell et al, Eds.: Springer-Verlag, pp. 173-188, 1995.

[Fei85] J. Feigenbaum, "*Encrypting Problem Instances: Or...Can You Take Advantage of Someone without Having to Trust Him?*", In Proc. Advances in Cryptology (Crypto'85), LNCS vol. 218, pp. 477 - 488, Springer-Verlag, 1986.

[Fel87] P. Feldman, "*A Practical Scheme for Non-interactive Verifiable Secret Sharing*", in Proc. 28th Annual. Symposium on the Foundations of Computer Science, pp. 427-437, IEEE, Oct. 12-14, 1987.

[Fel04] E. W. Felten, "*Report from Crypto 2004*", August 18, 2004, available at: http://www.freedom-to-tinker.com/archives/000664.html.

[FGM+96] Y. Frankel, P. Gemmell, P. D. MacKenzie, and M. Yung, "*Proactive RSA*", in Proc. Advances in Cryptology (Crypto'97), LNCS vol. 1294, Springer-Verlag, 1997.

[FPV98] A. Fuggetta, G. P. Picco, and G. Vigna, "*Understanding Code Mobility*", IEEE Transactions on Software Engineering, Vol. 24, 1998.

[FS96] S. Forrest and A. Soma, "*Building Diverse Computer Systems*", in Proc. of the 1996 Hot Topics of Operating Systems,

[Gar95] S. Garfinkel, "PGP: Pretty Good Privacy", O'Reilly & Assoc, 1995.

[GB99] S. Goldwasser and M. Bellare, "*Lecture Notes on Cryptography*", 1999, available at http://www-cse.ucsd.edu/users/mihir.

[GGM98] Y. Gertner, S. Goldwasser, and T. Malkin, "*A Random Server Model for Private Information Retrieval (or How to Achieve Information Theoretic PIR Avoiding Database Replication)*", in Proc. 2nd International Workshop on Randomization and Approximation Techniques in Computer Science, 1998.

[GIK+98] Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin, "*Protecting Data Privacy in Private Information Retrieval Schemes*", in Proc. of the 30th Annual ACM Symposium on the Theory of Computing, 1998, pp.151-160.

[GM84] S. Goldwasser and S. Micali, "*Probabilistic Encryption*", J. of Computer and Systems Sciences, vol. 28, 1984, pp. 270-299.

[Gol02] I. Goldberg, "*Privacy-Enhancing Technologies for the Internet, II: Five Years Later*", 2nd International Workshop on Privacy Enhancing Technologies (PET 2002), San Francisco, CA, USA, April 2002.

[Gon97] L. Gong, "*Survivable Mobile Code is Hard to Build*", in Proc. DARPA Workshop on Foundations for Secure Mobile Code Workshop, 26-28, March, Monterey, California, USA, 1997, available at: http://www.cs.nps.navy.mil/research/languages/wkshp.html.

[GRS99] D. Goldschlag, M. Reed, and P. Syverson, "*Onion Routing for Anonymous and Private Internet Connection*", Communication of the ACM, 42(2): 39-41, 1999.

[GWB97] I. Goldberg, D. Wagner, and E. Brewer, "*Privacy-Enhancing Technologies for the Internet*", in Proce. COMPCON'97, 1997.

[HJJ+97] A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk, and M. Yung. "*Proactive Public Key and Signature Systems*", in Proc. ACM Conference on Computer and Communications Security (CCS), 1997.

[HK93] Y. Huang and C. Kintala, "*Software Implemented Fault Tolerance: Technologies and Experience*", in Digest 23rd Annual International Symposium on Fault-Tolerant Computing, Toulouse, France, pp. 2-9, June, 1993.

[HLMR74] J. J. Horning, H. C. Lauer, P. M. Melliar-Smith and B. Randell, "*A Program Structure for Error Detection and Recovery*", Lecture Notes in Computer Science, 16:177-193, 1974.

[HW92] W. L. Heimerdinger and C. B. Weinstock, "*A Conceptual Framework for System Fault Tolerance*", Technical Report, CMU/SEI-92-TR-033, ESC-TR-92-033, Oct. 1992.

[ICG+98] A. Iyengar, R. Cahn, J. A. Garay, and C. Jutla, "*Design and Implementation of a Secure Distributed Data Repository*", in Proc. 14th IFIP International Information Security Conference (SEC 98), 1998.

[IK99] Y. Ishai and E. Kushilevitz, "*Improved Upper Bounds on Information Theoretic Private Information Retrieval*", in Proc 31st STOC, pp.79-88, 1999.

[IS03] A. Iliev and S. Smith, "*Privacy-Enhanced Credential Services*", in Proc. 2nd PKI Research Workshop, USA, 2003.

[IS04] A. Iliev and S.W. Smith, "*Private Information Storage with Logarithmic-space Secure Hardware*", in Proc. i-NetSec 04: 3rd Working Conference on Privacy and Anonymity in Networked and Distributed Systems, Kluwer, 2004, available at: http://www.cs.dartmouth.edu/~sws/abstracts/is04.shtml.

[ISN87] M. Ito, A. Saito and T. Nishizeki, "*Secret Sharing Scheme Realizing General Access Structure*", in Proc. Global Communication, 1987.

[ISO8895] ISO/IEC 8859, "*8-bit single-byte coded graphic character sets, Part 16: Latin alphabet No. 10*", published July 15, 2001, available at http://www.iso.ch/iso/en.

[Jar95] S. Jarecki, "*Proactive Secret Sharing and Public Key Cryptosystems*", Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology (MIT), 1995.

[Jai91] R. Jain, "*The Art of Computer Systems Performance Analysis, Techniques for Experimental Design, measurement, Simulation, and Modelling*", John Wiley & Sons, 1991, ISBN 0-471-50336-3.

[JDBC04] Sun Microsystem Ltd, JDBC technology, URL: http://java.sun.com/products/jdbc/, 2004.

[Ker88] B. W. Kernighan, "*The C Programming Language*", 2nd Edition, Prentice Hall PTR, 1988, ISBN: 0-13-110362-8.

[KG02] A. Kalagnanam and Balu G, "*Merlin Brings Nonblocking I/O to the Java Platform - New Addition Greatly Reduces Thread Overhead*", available at: http://www-106.ibm.com/developerworks/java/library/j-javaio/, 2002.

[Kim95] K. H. Kim, "*The Distributed Recovery Block Scheme*", in M. Lyu Eds., Software Fault Tolerance, John Wiley & Sons, 1995.

[KO97] E. Kushilevitz and R. Ostrovsky, "*Replication is Not Needed: Single Database, Computationally-Private Information Retrieval*", Proc. 38th Ann. IEEE Symposium on Foundations of Computer Science (FOCS'97), 1997, pp. 364-373.

[KR01] J. F. Kurose and W. R. Ross, "*Computer Networking: A Top-Down Approach Featuring the Internet*", Addison Wesley Logman, Inc., 2001.

[Kus03] E. Kushilevitz, "*Querying Databases Privately*", a presentation to IBM Almaden Research Lab, 2003, URL: http://www.almaden.ibm.com/institute/pdf/2003/EyalKushilevitz.pdf.

[LABKH87] J. C. Laprie, J. Arlat, C. Béounes, K. Kanoun and C. Hourtolle, "*Hardware and Software Fault Tolerance: Definition and Analysis of Architectural Solutions*", in Digest of 17th FTCS, pages 116-121, Pittsburgh, PA, 1987.

[Lap92] J.-C. Laprie (Ed.), "*Dependability: Basic Concepts and Terminology*", Dependable Computing and Fault-Tolerance, Springer-Verlag, Vienna, Austria, 1992.

[Lap95] J.-C. Laprie (Ed.), "*Dependable Computing: Concepts, Limits, Challenges*", in Special Issue, 25th IEEE International Symposium on Fault-Tolerant Computing (FTCS-25), Pasadena, CA, USA, pp. 42-54, IEEE Computer Society Press, 1995.

[LN83] R. Lidl and H. Niederreiter, Finite Fields, Encyclopaedia of Mathematics and Its Applications (vol. 20), Addison-Wesley, Reading, 1983.

[LN97] R. Lidl and H. Niederreiter, Finite Fields, Encyclopaedia of Mathematics and Its Applications (vol. 20), 2nd Edition, Cambridge University Press, 1997, reprinted 2000.

[LP84] R. Lidl and G. Pilz, Applied Abstract Algebra, Undergraduate Texts in Mathematics, Springer-Verlag, New York, 1984.

[Mal00] T. G. Malkin, "*A Study of Secure Database Access and General Two-Party Computation*", Ph.D. Thesis, MIT, Feb. 2000.

[McC04] D. McCullagh, "*Crypto Researchers Abuzz over Flaws*", available at: http://news.com.com/Crypto+researchers+abuzz+over+flaws/2100-1002_3-5313655.html.

[MC00] U. Moller and L. Cottrell, "*Mixmaster Protocol*", version 2, draft, June 2000, available at: http://www.eskimo.com/~rowdenw/crypt/Mix/.

[MKKW99] D. Mazieres, M. Kaminsky, M. F. Kaashoek, and E. Witchel. "*Separating Key Management from File System Security*", in Proc. 17th ACM SOSP, Kiawah Island, SC, Dec. 1999.

[MOV97] A. Menezes, P. van Oorschot, and S. Vanstone, Handbook of Applied Cryptography, CRC Press, 1997, available at http://www.cacr.math.uwaterloo.ca/hac.

[MS81] R. J. McEliece and D. V. Sarwate, "*On Sharing Secrets and Reed-Solomon Codes*", CACM, vol. 24, no. 9, 1981.

[Mul93] Sape Muller edits, Distributed Systems, Addison-Wesley, 1993.

[MySQL04] MySQL.com, URL: www.mysql.com, 2004.

[Nee89] R. M. Needham, "*Using Cryptography for Authentication*", Chapter 6, Sape Mullender Ed., Distributed Systems, ACM Press, Addison-Wesley, 1989, pp. 103 - 116.

[P3P01] The Platform for Privacy Preferences 1.0 (P3P 1.0) Specification, W3C, Sept. 2001.

[PS01] MAFTIA deliverable D2, "*Project IST-1999-11583 Malicious- and Accidental-Fault Tolerance for Internet Applications – Conceptual Model and Architecture*", David Powell and Robert Stroud (Eds.), 2001.

[PG04] D. Pinkas and T. Gindin, "*Internet X.509 Public Key Infrastructure Permanent Identifier*", IETF Internet draft <draft-ietf-pkix-pi-09.txt>, issued January 2004, expires July 2004, PKIX Working Group, available at http://www.ieft.org/internet-drafts/draft-ietf-pkix-pi-09.txt, accessed at June 2004.

[RAD78] R. L. Rivest, L. Adlerman, and M. L. Dertouzos, "*On Data Banks and Privacy Homomorphisms*", in Foundations of Secure Computation, Academic Press, pp. 171-179, 1978.

[Rab89] M. O. Rabin, "*Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance*", Journal of the ACM, 36(2): 335-348, 1989.

[Rew04] Rewebber, an online tool for providing private web surfing services, www.rewebber.com.

[RFL+96] M. K. Reiter, M. K. Franklin, J. B. Lacy, and R. N. Wright, "*The Omega Key Management Services*", in Journal of Computer Security, vol. 4, pp. 267-287, 1996.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman, "*A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*", Communication of the ACM, 21(2), pp. 120-126, 1978.

[RR98] M. K. Reiter and A. D. Rubin, "*Crowds: Anonymity for Web Transactions*", ACM Transactions on Information System Security, 1(1), Apr. 1998.

[RS60] I. S. Reed and G. Solomon, "*Polynomial Codes over Certain Finite Fields*", Journal of the SIAM, 8(2): 300-304, June 1960.

[Sch90] F. B. Schneider, "*Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial*", ACM Computing Surveys, 22(4): 299-319, Dec. 1990.

[Sch93] Fred B. Schneider, "*What Good are Models and What Models are Good?*", a chapter in the 2nd edition of "Distributed Systems" edited by Sape Mullender, Addison-Wesley, 1993, pp. 17-26.

[Sha79] A. Shamir, "*How to Share a Secret*", Communications of the ACM, 22(11):612-613, 1979.

[Shi04] R. J. Shimonshki, "*Wireless Attacks Primer*", Windows Security.com, accessed at Sept. 2004, available at: http://www.windowssecurity.com/articles/Wireless_Attack_Primer.html.

[SS00] S. W. Smith and D. Safford, "Practical Private Information Retrieval with Secure Co-processors", Technical Report, IBM Research Division, T. J. Watson Researcher Centre, July, 2000.

[SS01] S. W. Smith and D. Safford, "*Practical Server Privacy with Secure Co-processors*", IBM System Journal, 40(3), Sept. 2001.

[ST98a] T. Sander and C. F. Tschudin, "*Protecting Mobile Agents against Malicious Hosts*", Mobile Agents and Security, (G. Vigna, ed.), Lecture Notes on Computer Science 1419, 1998.

[ST98b] T. Sander and C. F. Tschudin, "*Towards Mobile Cryptography*", in Proc. IEEE Symposium on Security and Privacy, 1998.

[Ste98] W. Richard Stevens, "Unix Network Programming – Networking APIs: Sockets and XTI", Volume 1, 2nd Edition, Prentice Hall PTR, 1998, ISBN: 0-13-490012-X.

[Ste94] W. Richard Stevens, "*TCP/IP Illustrated Volume 1 – The Protocols*", Addison-Wesley Professional Computing Series, Addison-Wesley, 1994, ISBN: 0201633469.

[TW88] M. Tompa and H. Woll, "*How to Share a Secret with Cheaters*", J. of Cryptography, vol. 1, no. 2, 1988, pp. 133-138.

[TMB80] D. J. Taylor, D. E. Morgan, and J. P. Black, "*Redundancy in Data Structures: Improving Software Fault Tolerance*", IEEE Transactions in Software Engineering, vol. 6, pp. 585 - 593, 1980.

[VNC03] P. E. Verissimo, N. F. Neves, and M. P. Correia, "*Intrusion-Tolerant Architectures: Concepts and Design*", in a Book – Architecting Dependable Systems, R. Lemos, C. Gacek, A. Romanovsky (eds.), LNCS2677, Springer Verlag, 2003, also available at: http://www.navigators.di.fc.ul.pt.

[VS97] R. Ostrovsky and V. Shoup, "*Private Information Storage*", in Proc. 29th Annual ACM Symposium on Theory of Computing (STOC), pp. 294-303, 1997.

[Xu04] J. Xu, "*Towards Dependable and Secure e-Science/GRID Applications*", invited talk given at the open ceremony of e-Science building in the University of Newcastle, 11th May 2004.

[YXB02a] E. Y. Yang, J. Xu and K. H. Bennett, "*Private Information Retrieval in the Presence of Malicious Faults*", in Proc. 26th IEEE International Symposium on Computer Software and Applications (COMPSAC 2002), Oxford, UK, Aug. 2002, pp. 805-810.

[YXB02b] E. Y. Yang, J. Xu and K. H. Bennett, "*A Fault-Tolerant Approach to Secure Information Retrieval*", in Proc. 21st IEEE International Symposium on Reliable Distributed Systems (SRDS2002), Suita, Osaka, Japan, Oct. 2002, pp. 12-21.

[YXB03] E. Y. Yang, J. Xu, K. H. Bennett, "*Sharing with Limited Trust: An Attack-Tolerant Service in Durham e-Demand Project*", in Proc. UK e-Science Second All-Hands Meeting, Simon J. Cox Eds, Nottingham Conference Centre, U.K., Sept. 2nd-4th, 2003, ISBN 1-904425-11-9.

[Wai89] M. Waidner, "*Unconditional Sender and Recipient Untraceability in Spite of Active Attacks*", in Advances in Cryptology – Eurocrypt' 89, Lecture Notes in Computer Science, Vol. 434, Spriner-Verlag, 1989.

[WDHK01] C. Wang, J. Davidson, J. Hill, J. Knight, "*Protection of Software-based Survivability Mechanisms*", in Proc. Dependable Systems and Networks (DSN'01), 2001.

[Wel02] M. Welsh, "*NBIO: Nonblocking I/O for Java*", available at http://www.eecs.harvard.edu/~mdw/proj/java-nbio/, 2002.

[WLF+04] X. Wang, X. Lai, D. Feng, and H. Yu, "*Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD*", Crypto'04 Rump Sessions, 2004.

[WMB99] T. Wu, M. Malkin, and D. Boneh, "*Building Intrusion Tolerant Applications*", in Proc. 8th USENIX Security Symposium, Washington D.C., USA, 1999.

[WRC00] M. Waldman, A. D. Rubin, and L. F. Cranor, "*Publius: a Robust, Tamper-Evident, Censorship-resistant, Web Publishing System*", in Proc. 9th USENIX Security Symposium, Denver, CO, USA, 2000.

[WW04] S. Wehner and R. de Wolf, "*Improved Lower Bounds for Locally Decodable Codes and Private Information Retrieval*", Quantum Physics, quant-ph/0403140, available at: http://arxiv.org/abs/quant-ph/0403140.

[ZGZ03] X. Zhang, R. Gupta, and Y. Zhang, "*Precise Dynamic Slicing Algorithms*", in Proc. IEEE/ACM International Conference on Software Engineering (ICSE), Portland, Oregon, May 2003.

[Zho01] L. Zhou, "Towards Fault-Tolerant and Secure On-line Services", Ph.D. dissertation, Dept. Computer Science, Cornell University, Ithaca, N.Y., USA, May 2001.

[ZSR02] L. Zhou, F. B. Schneider, and R. van Renesse, "*COCA: A Secure Distributed On-line Certification Authority*", ACM Transaction on Computer Systems, Vol. 20, No. 4, pp. 329 – 268, Nov. 2002.