PRIVACY AND INTEGRITY PRESERVING COMPUTATION IN DISTRIBUTED SYSTEMS

By

Fei Chen

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Computer Science

2011

ABSTRACT

PRIVACY AND INTEGRITY PRESERVING COMPUTATION IN DISTRIBUTED SYSTEMS

By

Fei Chen

Preserving privacy and integrity of private data has become core requirements for many distributed systems across different parties. In these systems, one party may try to compute or aggregate useful information from the private data of other parties. However, this party is not be fully trusted by other parties. Therefore, it is important to design security protocols for preserving such private data. Furthermore, one party may want to query the useful information computed from such private data. However, query results may be modified by a malicious party. Thus, it is important to design query protocols such that query result integrity can be verified.

In this dissertation, we study four important privacy and integrity preserving problems for different distributed systems. For two-tiered sensor networks, where storage nodes serve as an intermediate tier between sensors and a sink for storing data and processing queries, we proposed SafeQ, a protocol that prevents compromised storage nodes from gaining information from both sensor collected data and sink issued queries, while it still allows storage nodes to process queries over encrypted data and the sink to detect compromised storage nodes when they misbehave. For cloud computing, where a cloud provider hosts the data of an organization and replies query results to the customers of the organization, we propose novel privacy and integrity preserving schemes for multi-dimensional range queries such that the cloud provider can process encoded queries over encoded data without knowing the actual

values, and customers can verify the integrity of query results with high probability. For distributed firewall policies, we proposed the first privacy-preserving protocol for cross-domain firewall policy optimization. For any two adjacent firewalls belonging to two different administrative domains, our protocol can identify in each firewall the rules that can be removed because of the other firewall. For network reachability, one of the key factors for capturing end-to-end network behavior and detecting the violation of security policies, we proposed the first cross-domain privacy-preserving protocol for quantifying network reachability.

ACKNOWLEDGMENTS

I am extremely grateful to my advisor Dr. Alex X. Liu. He is not only an excellent advisor, an outstanding researcher, but also a great friend. This thesis would not be possible without his tremendous help. He always tries his best to guide me through every perspective of my graduate study, and give tremendous support to make me successful in both study and research. He not only teaches me how to identify problems, how to solve problems, and how to build systems, but also helps me improve my writing skills, speaking skills, and communication skills. Numerous times, he helps me set milestones together, and guides me to make progress steadily. When I meet difficult problems in my research, his encouragement help me to gain more confidence, and to come up with solutions.

I would like to thank other members in my thesis committee, Dr. Eric Torng, Dr. Richard Enbody, and Dr. Hayder Radha. Dr. Torng not only guided my thesis, but also gave me great help on finding travel funds such that I can present my papers in many conferences. Dr. Enbody and Dr. Radha gave me many valuable suggestions and feedbacks on my qualifying report and comprehensive proposal, which helped improve my thesis significantly.

I thank my collaborator and friend Bezawada Bruhadeshwar. He makes significant contributions to this thesis work. My work with him is both happy and fruitful. He actively discussed with me the works of privacy preserving firewall optimization and privacy preserving network qualifications and provide significant help to move the project forward.

I really thank my wife Xiaoshu Wu and my parents for her great love and tremendous support in every aspect of my life in Michigan State University. They support me to pursue my own goals with confidence and to overcome obstacles with courage. I extremely grateful for my wife taking great care of me when I had serious health problems and stayed in hospital. This thesis is dedicated to my wife and my parents.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

For distributed systems across different parties, preserving privacy and integrity of private data has become core requirements in the recent decade. In these systems, one party may not be fully trusted by other parties, but tries to compute or aggregate useful information from private data of other parties. Thus, it is very important to design security communication and storage protocols for preventing the party from revealing such data of other parties. Otherwise, one party would be reluctant to share its private data. For example, Google and Facebook collect significant amount of personal data every day through the Internet, while many persons want to preserve the privacy of such data due to the security concern. Furthermore, one party may want to query the useful information computed from the private data of other parties. However, query results may be modified by a malicious party. Thus, it is very important to design query protocols such that the integrity of query results can be verified.

In this dissertation, we study four important, yet under-investigated, privacy and integrity preserving problems for different distributed systems, privacy and integrity preserving range queries in sensor networks, privacy and integrity preserving range queries for cloud computing, privacy preserving cross-domain cooperative firewall optimization, and privacy pre-

serving cross-domain network reachability quantification. Next, for each problem, we first describe the motivation, present the challenges, and propose our solution.

1.1 Privacy and Integrity Preserving Queries for Sensor Networks

Two-tiered sensor networks, where storage nodes gather data from nearby sensors and answer queries from the sink, has been widely adopted due to the benefits of power and storage saving for sensors and the efficiency of query processing. However, a compromised storage node could disclose all sensitive data collected from nearby sensors and return forged data to the sink. Therefore, on a storage node, the data received from sensors should have been encrypted and the queries received from the sink should also have been encrypted.

Without decrypting the data and queries, the storage node needs to process encrypted queries over encrypted data correctly and send encrypted data that satisfy the queries back to the sink. Moreover, the sink needs to be able to verify the integrity of the query results received from storage nodes. Seemingly impossible, this problem is very challenging to solve. It is even more challenging to solve efficiently.

To preserve privacy, I proposed a novel technique to encode both data and queries such that a storage node can correctly process encoded queries over encoded data without knowing their values. To preserve integrity, I proposed two schemes, one using Merkle hash trees and another using a new data structure called neighborhood chains, to generate integrity verification information so that a sink can use this information to verify whether the result of a query contains exactly the data items that satisfy the query. To improve performance, I proposed an optimization technique using Bloom filters to reduce the communication cost between sensors and storage nodes.

1.2 Privacy and Integrity Preserving Queries for Cloud Computing

Outsourced database systems are one of the most important work in cloud computing, where a cloud provider hosts the private databases of an organization and replies query results to the customers on behalf of the organization. However, the inclusion of the outsourced database systems also brings significant security and privacy challenges. As cloud providers cannot be fully trusted and the data of an organization are typically confidential, the organization always encodes the data before storing them in a cloud to prevent the cloud provider from revealing the data.

However, it is difficult to process queries over encoded data. Furthermore, since cloud providers serve as an important role for answering queries from customers, they may misbehave the query results, such as returning forged data for the query or not returning all data that satisfy the query. I proposed the basic idea of the privacy and integrity preserving protocol for processing range queries in outsourced databases.

To preserve privacy, I proposed an order preserving hash function to encode the data items from the data owner and the queries from its customers such that the cloud provider can use the encoded queries and encoded data items to find out the query results without knowing the actual values. To preserve integrity, I propose the first probabilistic integrity preserving scheme for range queries in outsourced database systems. This scheme allows a customer to verify the integrity of a query result with a high probability.

1.3 Privacy Preserving Optimization for Firewall Policies

Firewalls have been widely deployed on the Internet for securing private networks by checking whether to accept or discard packets based on its policy. Optimizing firewall policies is crucial for improving network performance. Prior work on firewall optimization focuses on either intra-firewall or inter-firewall optimization within one administrative domain where the privacy of firewall policies is not a concern. I explored inter-firewall optimization across administrative domains for the first time.

The key technical challenge is that firewall policies cannot be shared across domains because a firewall policy contains confidential information and even potential security holes, which can be exploited by attackers.

In this work, I proposed the first cross-domain privacy-preserving cooperative firewall policy optimization protocol. Specifically, for any two adjacent firewalls belonging to two different administrative domains, the protocol can identify in each firewall the rules that can be removed because of the other firewall. The optimization process involves cooperative computation between the two firewalls without any party disclosing its policy to the other.

1.4 Privacy Preserving Quantification of Network Reachability

Network reachability is one of the key factors for capturing end-to-end network behavior and detecting the violation of security policies. Many approaches were proposed to address the network reachability problem. The main assumption in all these approaches is that the reachability restriction information of each network device and other configuration state is

known to a central network analyst, who is quantifying the network reachability.

However, in reality, it is common that the network devices along a given path belong to different administrative domains where the reachability restriction information cannot be shared with others including the network analyst.

In this work, I will try to design the first cross-domain privacy-preserving protocol for quantifying network reachability. The protocol enables the network analyst to accurately determine the network reachability along a network path through different administrative domains without knowing the the reachability restriction information of the other domains.

1.5 Structure of the paper

In Chapter 2, we present SafeQ, a protocol that prevents attackers from gaining information from both sensor collected data and sink issued queries. we start with the system model, threat model, and problem statement. Then, we present our schemes for preserving data privacy and query result integrity, respectively. We also propose a solution to adapt SafeQ for event-driven sensor networks. We show that SafeQ excels state-of-the-art scheme in both privacy and performance.

In Chapter 3, we first propose an efficient privacy-preserving scheme that can process multi-dimensional range queries without false positives. Then, we propose the first probabilistic scheme for verifying the integrity of range query results. This scheme employs a new data structure, *local bit matrices*, which enables customers to verify query result integrity with high probability. We show that our scheme is effectiveness and efficiency on both real and synthetic datasets.

In Chapter 4, we first introduce the problem, system model, and threat model. Then, we present our privacy-preserving protocol for detecting inter-firewall redundant rules. Finally, we give security analysis results of our protocol and present our experimental results.

In Chapter 5, we first propose the cross-domain privacy-preserving protocol to quantify network reachability across multiple parties. Then, we propose an optimization technique to reduce computation and communication costs. Finally, we show that our protocol is efficient and suitable for real applications.

In Chapter 6, we first survey related work of secure multiparty computation, one of the fundamental cryptographic primitives for designing privacy-preserving protocols. Then, we discuss related work for each specific problem we investigate. in two parts.

Finally, in Chapter 7, we summarize this dissertation, discuss limitations, and outline future research directions.

CHAPTER 2

Privacy and Integrity Preserving Range Queries in Sensor Networks

2.1 Introduction

2.1.1 Motivation

Wireless sensor networks have been widely deployed for various applications, such as environment sensing, building safety monitoring, and earthquake predication, *etc.*. In this work, we consider a two-tiered sensor network architecture in which storage nodes gather data from nearby sensors and answer queries from the sink of the network. The storage nodes serve as an intermediate tier between the sensors and the sink for storing data and processing queries. Storage nodes bring three main benefits to sensor networks. First, sensors save power by sending all collected data to their closest storage node instead of sending them to the sink through long routes. Second, sensors can be memory limited because data are mainly stored on storage nodes. Third, query processing becomes more efficient because the sink only communicates with storage nodes for queries. The inclusion of storage nodes in sensor networks

was first introduced in [65] and has been widely adopted [26, 82, 70, 71, 69]. Several products of storage nodes, such as StarGate [7] and RISE [6], are commercially available.

However, the inclusion of storage nodes also brings significant security challenges. As storage nodes store data received from sensors and serve as an important role for answering queries, they are more vulnerable to be compromised, especially in a hostile environment. A compromised storage node imposes significant threats to a sensor network. First, the attacker may obtain sensitive data that has been, or will be, stored in the storage node. Second, the compromised storage node may return forged data for a query. Third, this storage node may not include all data items that satisfy the query.

Therefore, we want to design a protocol that prevents attackers from gaining information from both sensor collected data and sink issued queries, which typically can be modeled as range queries, and allows the sink to detect compromised storage nodes when they misbehave. For privacy, compromising a storage node should not allow the attacker to obtain the sensitive information that has been, and will be, stored in the node, as well as the queries that the storage node has received, and will receive. Note that we treat the queries from the sink as confidential because such queries may leak critical information about query issuers' interests, which need to be protected especially in military applications. For integrity, the sink needs to detect whether a query result from a storage node includes forged data items or does not include all the data that satisfy the query.

2.1.2 Technical Challenges

There are two key challenges in solving the privacy and integrity preserving range query problem. First, a storage node needs to correctly process encoded queries over encoded data without knowing their actual values. Second, a sink needs to verify that the result of a query contains all the data items that satisfy the query and does not contain any forged data.

2.1.3 Limitations of Prior Art

Although important, the privacy and integrity preserving range query problem has been under-investigated. The prior art solution to this problem was proposed by Sheng and Li in their recent seminal work [69]. We call it S&L scheme. This scheme has two main drawbacks: (1) it allows attackers to obtain a reasonable estimation on both sensor collected data and sink issued queries, and (2) the power consumption and storage space for both sensors and storage nodes grow exponentially with the number of dimensions of collected data.

2.1.4 Our Approach and Key Contributions

In this work, we propose SafeQ, a novel privacy and integrity preserving range query protocol for two-tiered sensor networks. The ideas of SafeQ are fundamentally different from S&L scheme. To preserve privacy, SafeQ uses a novel technique to encode both data and queries such that a storage node can correctly process encoded queries over encoded data without knowing their actual values. To preserve integrity, we propose two schemes, one using Merkle hash trees and another using a new data structure called neighborhood chains, to generate integrity verification information such that a sink can use this information to verify whether the result of a query contains exactly the data items that satisfy the query. We also propose an optimization technique using Bloom filters to significantly reduce the communication cost between sensors and storage nodes. Furthermore, we propose a solution to adapt SafeQ for event-driven sensor networks, where a sensor submits data to its nearby storage node only when a certain event happens and the event may occur infrequently.

SafeQ excels state-of-the-art S&L scheme [69] in two aspects. First, SafeQ provides significantly better security and privacy. While prior art allows a compromised storage node to obtain a reasonable estimation on the value of sensor collected data and sink issued queries, SafeQ makes such estimation very difficult. Second, SafeQ delivers orders of magnitude bet-

ter performance on both power consumption and storage space for multi-dimensional data, which are most common in practice as most sensors are equipped with multiple sensing modules such as temperature, humidity, pressure, etc.

2.1.5 Summary of Experimental Results

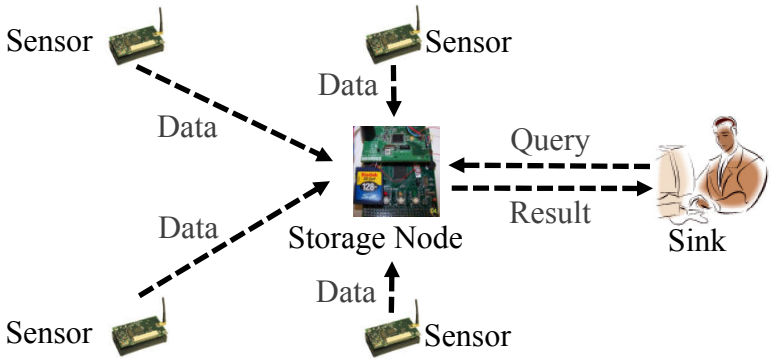
We performed side-by-side comparison with prior art over a large real-world data set from Intel Lab [4]. Our results show that the power and space savings of SafeQ over prior art grow exponentially with the number of dimensions. For power consumption, for three-dimensional data, SafeQ consumes 184.9 times less power for sensors and 76.8 times less power for storage nodes. For space consumption on storage nodes, for three-dimensional data, SafeQ uses 182.4 times less space. Our experimental results conform with the analysis that the power and space consumption in S&L scheme grow exponentially with the number of dimensions, whereas those in SafeQ grow linearly with the number of dimensions times the number of data items.

2.2 Models and Problem Statement

2.2.1 System Model

We consider two-tired sensor networks as illustrated in Figure 2.1. A two-tired sensor network consists of three types of nodes: *sensors*, *storage nodes*, and a *sink*. Sensors are inexpensive sensing devices with limited storage and computing power. They are often massively distributed in a field for collecting physical or environmental data, *e.g.*, temperature. Storage nodes are powerful mobile devices that are equipped with much more storage capacity and computing power than sensors. Each sensor periodically sends collected data to its nearby storage node. The sink is the point of contact for users of the sensor network. Each time the

sink receives a question from a user, it first translates the question into multiple queries and then disseminates the queries to the corresponding storage nodes, which process the queries based on their data and return the query results to the sink. The sink unifies the query results from multiple storage nodes into the final answer and sends it back to the user.



For interpretation of the references to color in this and all other figures, the reader is referred to the electronic version of this dissertation.

Figure 2.1. Architecture of two-tired sensor networks

For the above network architecture, we assume that all sensor nodes and storage nodes are loosely synchronized with the sink. With loose synchronization in place, we divide time into fixed duration intervals and every sensor collects data once per *time interval*. From a starting time that all sensors and the sink agree upon, every n time intervals form a *time slot*. From the same starting time, after a sensor collects data for n times, it sends a message that contains a 3-tuple $(i, t, \{d_1, \cdots, d_n\})$, where i is the sensor ID and t is the sequence number of the time slot in which the n data items $\{d_1, \cdots, d_n\}$ are collected by sensor s_i. We address privacy and integrity preserving ranges queries for event-driven sensor networks, where a sensor only submits data to a nearby storage node when a certain event happens, in Section 2.7. We further assume that the queries from the sink are range queries. A range query "finding all the data items, which are collected at time slot t and whose value is in the range $[a, b]$" is denoted as $\{t, [a, b]\}$. Note that the queries in most sensor network

applications can be easily modeled as range queries. For ease of presentation, Table 2.1 shows the notation used in this chapter.

s_i	A sensor with ID i
k_i	The secret key of sensor s_i
t	The sequence number of a time slot
$d_1, \cdots, d_n$	n 1-dimensional data items
$D_1, \cdots, D_n$	n z-dimensional data items
$\mathcal{H}, \mathcal{G}, \mathcal{E}$	Three "magic" functions
$\mathcal{F}(x)$	The prefix family of x
$\mathcal{S}([d_1, d_2])$	The minimum set of prefixes converted from $[d_1, d_2]$
$\mathcal{N}$	A prefix numericalization function
$HMAC_g$	An HMAC function with key g
QR	A query result
VO	An integrity verification object

Table 2.1. Summary of notation

2.2.2 Threat Model

For a two-tiered sensor network, we assume that the sensors and the sink are trusted but the storage nodes are not. In a hostile environment, both sensors and storage nodes can be compromised. If a sensor is compromised, the subsequent collected data of the sensor will be known to the attacker and the compromised sensor may send forged data to its closest storage node. It is extremely difficult to prevent such attacks without the use of tamper proof hardware. However, the data from one sensor constitute a small fraction of the collected data of the whole sensor network. Therefore, we mainly focus on the scenario where a storage node is compromised. Compromising a storage node can cause much greater damage to the sensor network than compromising a sensor. After a storage node is compromised, the large quantity of data stored on the node will be known to the attacker and upon receiving a query from the sink, the compromised storage node may return a falsified result formed by

including forged data or excluding legitimate data. Therefore, attackers are more motivated to compromise storage nodes.

2.2.3 Problem Statement

The fundamental problem for a two-tired sensor network is: *how can we design the storage scheme and the query protocol in a privacy and integrity preserving manner*? A satisfactory solution to this problem should meet the following two requirements: (1) *Data and query privacy*: Data privacy means that a storage node cannot know the actual values of sensor collected data. This ensures that an attacker cannot understand the data stored on a compromised storage node. Query privacy means that a storage node cannot know the actual value of sink issued queries. This ensures that an attacker cannot understand, or deduce useful information from, the queries that a compromised storage node receives. (2) *Data integrity*: If a query result that a storage node sends to the sink includes forged data or excludes legitimate data, the query result is guaranteed to be detected by the sink as invalid. Besides these two hard requirements, a desirable solution should have low power and space consumption because these mobile devices have limited resources.

2.3 Privacy for 1-dimensional Data

To preserve privacy, it seems natural to have sensors encrypt data and the sink encrypt queries; however, the key challenge is how a storage node processes encrypted queries over encrypted data.

The idea of our solution for preserving privacy is illustrated in Figure 2.2. We assume that each sensor s_i in a network shares a secret key k_i with the sink. For the n data items $d_1, \cdots, d_n$ that a sensor s_i collects in time slot t, s_i first encrypts the data items using key k_i, the results of which are represented as $(d_1)_{k_i}, \cdots, (d_n)_{k_i}$. Then, s_i applies a

"magic" function $\mathcal{H}$ to the n data items and obtains $\mathcal{H}(d_1, \cdots, d_n)$. The message that the sensor sends to its closest storage node includes both the encrypted data and the associative information $\mathcal{H}(d_1, \cdots, d_n)$. When the sink wants to perform query $\{t, [a, b]\}$ on a storage node, the sink applies another "magic" function $\mathcal{G}$ on the range $[a, b]$ and sends $\{t, \mathcal{G}([a, b])\}$ to the storage node. The storage node processes the query $\{t, \mathcal{G}([a, b])\}$ over encrypted data $(d_1)_{k_i}, \cdots, (d_n)_{k_i}$ collected at time slot t using another "magic" function $\mathcal{E}$. The three "magic" functions $\mathcal{H}$, $\mathcal{G}$, and $\mathcal{E}$ satisfy the following three conditions: (1) A data item d_j $(1 \leq j \leq n)$ is in range $[a, b]$ if and only if $\mathcal{E}(j, \mathcal{H}(d_1, \cdots, d_n), \mathcal{G}([a, b]))$ is true. This condition allows the storage node to decide whether $(d_j)_{k_i}$ should be included in the query result. (2) Given $\mathcal{H}(d_1, \cdots, d_n)$ and $(d_j)_{k_i}$, it is computationally infeasible for the storage node to compute d_j. This condition guarantees data privacy. (3) Given $\mathcal{G}([a, b])$, it is computationally infeasible for the storage node to compute $[a, b]$. This condition guarantees query privacy.

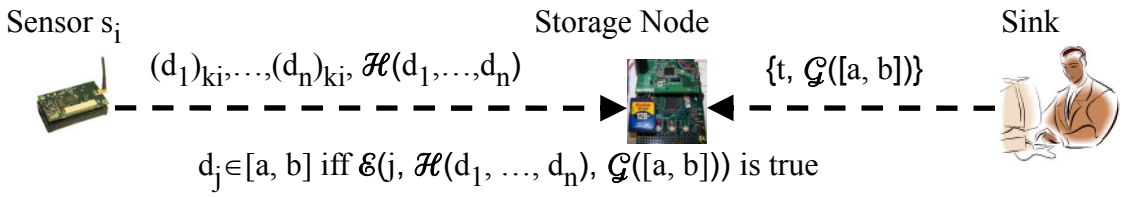


Figure 2.2. The idea of SafeQ for preserving privacy

2.3.1 Prefix Membership Verification

The building block of our privacy preserving scheme is the prefix membership verification scheme first introduced in [23] and later formalized in [48]. The idea of this scheme is to convert the verification of whether a number is in a range to several verifications of whether two numbers are equal. A prefix $\{0, 1\}^k\{*\}^{w-k}$ with k leading 0s and 1s followed by $w - k$ $*$s is called a $k-$prefix. For example, $1***$ is a 1-prefix and it denotes the range $[1000, 1111]$.

If a value x matches a $k-$prefix (*i.e.*, x is in the range denoted by the prefix), the first k bits of x and the $k-$prefix are the same. For example, if $x \in 1***$ (*i.e.*, $x \in [1000, 1111]$), then the first bit of x must be 1. Given a binary number $b_1b_2\cdots b_w$ of w bits, the prefix family of this number is defined as the set of $w+1$ prefixes $\{b_1b_2\cdots b_w, b_1b_2\cdots b_{w-1}*, \cdots, b_1*\cdots*, **...*\}$, where the i-th prefix is $b_1b_2\cdots b_{w-i+1}*\cdots*$. The prefix family of x is denoted as $\mathcal{F}(x)$. For example, the prefix family of number 12 is $\mathcal{F}(12) = \mathcal{F}(1100) =$\{1100, 110*, 11**, 1***, ****\}. Prefix membership verification is based on the fact that for any number x and prefix P, $x \in P$ if and only if $P \in \mathcal{F}(x)$.

To verify whether a number a is in a range $[d_1, d_2]$, we first convert the range $[d_1, d_2]$ to a minimum set of prefixes, denoted $\mathcal{S}([d_1, d_2])$, such that the union of the prefixes is equal to $[d_1, d_2]$. For example, $\mathcal{S}([11, 15]) =$\{1011,11**\}. Given a range $[d_1, d_2]$, where d_1 and d_2 are two numbers of w bits, the number of prefixes in $\mathcal{S}([d_1, d_2])$ is at most $2w-2$ [39]. Second, we compute the prefix family $\mathcal{F}(a)$ for number a. Thus, $a \in [d_1, d_2]$ if and only if $\mathcal{F}(a) \cap \mathcal{S}([d_1, d_2]) \neq \emptyset$.

To verify whether $\mathcal{F}(a) \cap \mathcal{S}([d_1, d_2]) \neq \emptyset$ using only the operations of verifying whether two numbers are equal, we convert each prefix to a corresponding unique number using a prefix numericalization function. A prefix numericalization function $\mathcal{N}$ needs to satisfy the following two properties: (1) for any prefix P, $\mathcal{N}(P)$ is a binary string; (2) for any two prefixes P_1 and P_2, $P_1 = P_2$ if and only if $\mathcal{N}(P_1) = \mathcal{N}(P_2)$. There are many ways to do prefix numericalization. We use the prefix numericalization scheme defined in [20]. Given a prefix $b_1b_2\cdots b_k*\cdots*$ of w bits, we first insert 1 after b_k. The bit 1 represents a separator between $b_1b_2\cdots b_k$ and $*\cdots*$. Second, we replace every * by 0. Note that if there is no * in a prefix, we add 1 at the end of this prefix. For example, $11**$ is converted to 11100. Given a set of prefixes S, we use $\mathcal{N}(S)$ to denote the resulting set of numericalized prefixes. Therefore, $a \in [d_1, d_2]$ if and only if $\mathcal{N}(\mathcal{F}(a)) \cap \mathcal{N}(\mathcal{S}([d_1, d_2])) \neq \emptyset$. Figure 2.3 illustrates the process of verifying $12 \in [11, 15]$.

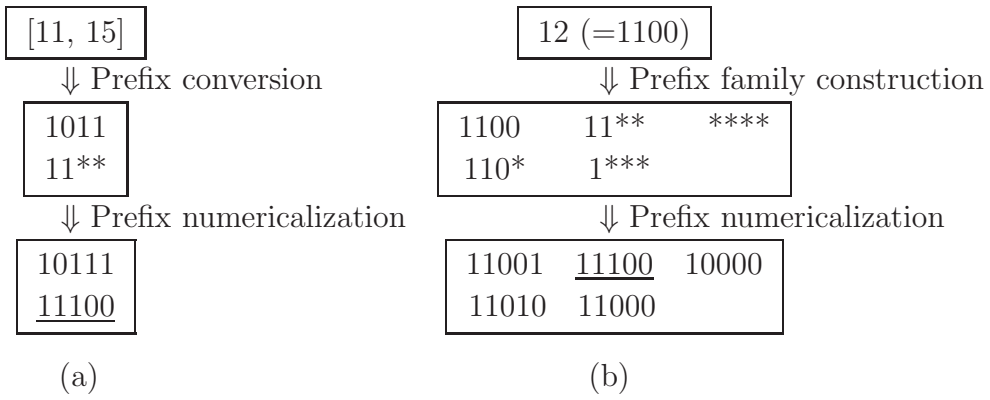


Figure 2.3. Prefix membership verification

2.3.2 The Submission Protocol

The submission protocol concerns how a sensor sends its data to a storage node. Let $d_1, \cdots, d_n$ be data items that sensor s_i collects at a time slot. Each item d_j $(1 \leq j \leq n)$ is in the range (d_0, d_{n+1}), where d_0 and d_{n+1} denote the lower and upper bounds, respectively, for all possible data items that a sensor may collect. The values of d_0 and d_{n+1} are known to both sensors and the sink. After collecting n data items, s_i performs six steps:

1) Sort the n data items in an ascending order. For simplicity, we assume $d_0 < d_1 < d_2 < \cdots < d_n < d_{n+1}$. If some data items have the same value, we simply represent them as one data item annotated with the number of such data items.

2) Convert the $n+1$ ranges $[d_0, d_1]$, $[d_1, d_2]$, $\cdots$, $[d_n, d_{n+1}]$ to their corresponding prefix representation, *i.e.*, compute $\mathcal{S}([d_0, d_1])$, $\mathcal{S}([d_1, d_2])$, $\cdots$, $\mathcal{S}([d_n, d_{n+1}])$.

3) Numericalize all prefixes. That is, compute $\mathcal{N}(\mathcal{S}([d_0, d_1]))$, $\cdots$, $\mathcal{N}(\mathcal{S}([d_n, d_{n+1}]))$.

4) Compute the keyed-Hash Message Authentication Code (HMAC) of each numericalized prefix using key g, which is known to all sensors and the sink. Examples of HMAC implementations include HMAC-MD5 and HMAC-SHA1 [46, 66, 29]. An HMAC function using key g, denoted $HMAC_g$, satisfies the one-wayness property (*i.e.*, given $HMAC_g(x)$,

it is computationally infeasible to compute x and g) and the collision resistance property (*i.e.*, it is computationally infeasible to find two distinct numbers x and y such that $HMAC_g(x) = HMAC_g(y)$). Given a set of numbers S, we use $HMAC_g(S)$ to denote the resulting set of numbers after applying function $HMAC_g$ to every number in S. In summary, this step computes $HMAC_g(\mathcal{N}(\mathcal{S}([d_0, d_1]))), \cdots, HMAC_g(\mathcal{N}(\mathcal{S}([d_n, d_{n+1}])))$.

5) Encrypt every data item with key k_i, *i.e.*, compute $(d_1)_{k_i}, \cdots, (d_n)_{k_i}$.

6) Sensor s_i sends the encrypted data along with $HMAC_g(\mathcal{N}(\mathcal{S}([d_0, d_1]))), \cdots, HMAC_g(\mathcal{N}(\mathcal{S}([d_n, d_{n+1}])))$ to its closest storage node.

The above steps show that the aforementioned "magic" function $\mathcal{H}$ is defined as follows:

$$\mathcal{H}(d_1, \cdots, d_n) = (HMAC_g(\mathcal{N}(\mathcal{S}([d_0, d_1]))), \cdots, HMAC_g(\mathcal{N}(\mathcal{S}([d_n, d_{n+1}]))))$$

Due to the one-wayness and collision resistance properties of the HMAC function, given $\mathcal{H}(d_1, \cdots, d_n)$ and the n encrypted data items $(d_1)_{k_i}, \cdots, (d_n)_{k_i}$, the storage node cannot compute the value of any data item.

2.3.3 The Query Protocol

The query protocol concerns how the sink sends a range query to a storage node. When the sink wants to perform query $\{t, [a, b]\}$ on a storage node, it performs the following four steps. Note that any range query $[a, b]$ satisfies the condition $d_0 < a \leq b < d_{n+1}$,

1) Compute prefix families $\mathcal{F}(a)$ and $\mathcal{F}(b)$.

2) Numericalize all prefixes, *i.e.*, compute $\mathcal{N}(\mathcal{F}(a))$, $\mathcal{N}(\mathcal{F}(b))$.

3) Apply $HMAC_g$ to each numericalized prefix, *i.e.*, compute $HMAC_g(\mathcal{N}(\mathcal{F}(a)))$ and $HMAC_g(\mathcal{N}(\mathcal{F}(b)))$.

4) Send $\{t, HMAC_g(\mathcal{N}(\mathcal{F}(a))), HMAC_g(\mathcal{N}(\mathcal{F}(b)))\}$ as a query to the storage node.

The above steps show that the aforementioned "magic" function $\mathcal{G}$ is defined as follows:

$$\mathcal{G}([a,b]) = (HMAC_g(\mathcal{N}(\mathcal{F}(a))), HMAC_g(\mathcal{N}(\mathcal{F}(b))))$$

Because of the one-wayness and collision resistance properties of the HMAC function, the storage node cannot compute a and b from the query that it receives.

2.3.4 Query Processing

Upon receiving query $\{t,\ HMAC_g(\mathcal{N}(\mathcal{F}(a))),\ HMAC_g(\mathcal{N}(\mathcal{F}(b)))\}$, the storage node processes this query on the n data items $(d_1)_{k_i}, \cdots, (d_n)_{k_i}$ received from each nearby sensor s_i at time slot t based on the following theorem.

Theorem 1. *Given n numbers sorted in the ascending order $d_1 < \cdots < d_n$, where $d_j \in (d_0, d_{n+1})$ $(1 \le j \le n)$, and a range $[a,b]$ $(d_0 < a \le b < d_{n+1})$, $d_j \in [a,b]$ if and only if there exist $1 \le n_1 \le j < n_2 \le n+1$ such that the following two conditions hold:*
(1) $HMAC_g(\mathcal{N}(\mathcal{F}(a))) \cap HMAC_g(\mathcal{N}(\mathcal{S}([d_{n_1-1}, d_{n_1}]))) \neq \emptyset$
(2) $HMAC_g(\mathcal{N}(\mathcal{F}(b))) \cap HMAC_g(\mathcal{N}(\mathcal{S}([d_{n_2-1}, d_{n_2}]))) \neq \emptyset$.

Proof. Note that $d_j \in [a,b]$ $(1 \le j \le n)$ if and only if there exist n_1 and n_2 $(1 \le n_1 \le j < n_2 \le n+1)$ such that $a \in [d_{n_1-1}, d_{n_1}]$ and $b \in [d_{n_2-1}, d_{n_2}]$. Further, $a \in [d_{n_1-1}, d_{n_1}]$ if and only if $HMAC_g(\mathcal{N}(\mathcal{F}(a))) \cap HMAC_g(\mathcal{N}(\mathcal{S}([d_{n_1-1}, d_{n_1}]))) \neq \emptyset$ and $b \in [d_{n_2-1}, d_{n_2}]$ if and only if $HMAC_g(\mathcal{N}(\mathcal{F}(b))) \cap HMAC_g(\mathcal{N}(\mathcal{S}([d_{n_2-1}, d_{n_2}]))) \neq \emptyset$ □

Based on Theorem 1, the storage node searches for the smallest n_1 and the largest n_2 $(1 \le n_1, n_2 \le n+1)$ such that $a \in [d_{n_1-1}, d_{n_1}]$ and $b \in [d_{n_2-1}, d_{n_2}]$. If $n_1 < n_2$, the data items $d_{n_1}, d_{n_1+1}, \cdots, d_{n_2-1}$ are in the range $[a,b]$; if $n_1 = n_2$, no data item is in the range $[a,b]$.

In fact, there is another privacy preserving scheme. First, sensor s_i converts each data value d_j to a prefix family $\mathcal{F}(d_j)$, and then applies the numericalization and hash functions $HMAC_g(\mathcal{N}(\mathcal{F}(d_j)))$. Second, the sink converts a given range query $[a, b]$ to a set of prefixes $\mathcal{S}([a, b])$, and then applies the numericalization and hash functions $HMAC_g(\mathcal{N}(\mathcal{S}([a, b])))$. Finally, the storage node checks whether $HMAC_g(\mathcal{N}(\mathcal{F}(d_j)))$ has a common element with $HMAC_g(\mathcal{N}(\mathcal{S}([a, b])))$. However, this privacy preserving scheme is not compatible with the integrity preserving scheme that we will discuss in Section 2.4. Because this privacy preserving scheme does not allow storage nodes to identify the positions of a and b (from the range query $[a, b]$) among $d_1, \cdots, d_n$ if no data item satisfies the query. While our integrity preserving scheme requires storage nodes to know such information in order to compute integrity verification objects.

2.4 Integrity for 1-dimensional Data

The meaning of data integrity is two-fold in this context. In the result that a storage node sends to the sink in responding to a query, first, the storage node cannot include any data item that does not satisfy the query; second, the storage node cannot exclude any data item that satisfies the query. To allow the sink to verify the integrity of a query result, the query response from a storage node to the sink consists of two parts: (1) the *query result* QR, which includes all the encrypted data items that satisfy the query; (2) the *verification object* VO, which includes information for the sink to verify the integrity of QR. To achieve this purpose, we propose two schemes based on two different techniques, Merkle hash trees and neighborhood chains.

2.4.1 Integrity Scheme Using Merkle Hash Trees

Our first integrity preserving mechanism is based on Merkle hash trees [58]. Each time a sensor sends data items to storage nodes, it constructs a Merkle hash tree for the data items. Figure 2.4 shows a Merkle hash tree constructed for eight data items. Suppose sensor s_i wants to send $n = 2^m$ encrypted data items $(d_1)_{k_i}, \cdots, (d_n)_{k_i}$ to a storage node. Sensor s_i first builds a Merkle hash tree for the $n = 2^m$ data items, which is a complete binary tree. The n terminal nodes are $H_1, \cdots, H_n$, where $H_j = h((d_j)_{k_i})$ for every $1 \leq j \leq n$. Function h is a one-way hash function such as MD5 [66] or SHA-1 [29]. The value of each nonterminal node v, whose children are v_l and v_r, is the hash of the concatenation of v_l's value and v_r's value. For example, in Figure 2.4, $H_{12} = h(H_1|H_2)$. Note that if the number of data items n is not a power of 2, interim hash values that do not have a sibling value to which they may be concatenated are promoted, without any change, up the tree until a sibling is found. Note that the resulting Merkle hash tree will not be balanced. For the example Merkle hash tree in Figure 2.4, if we remove the nodes H_6, H_7, H_8, H_{78}, and let $H_{58} = H_{56} = H_5$, the resulting unbalanced tree is the Merkle hash tree for 5 data items.

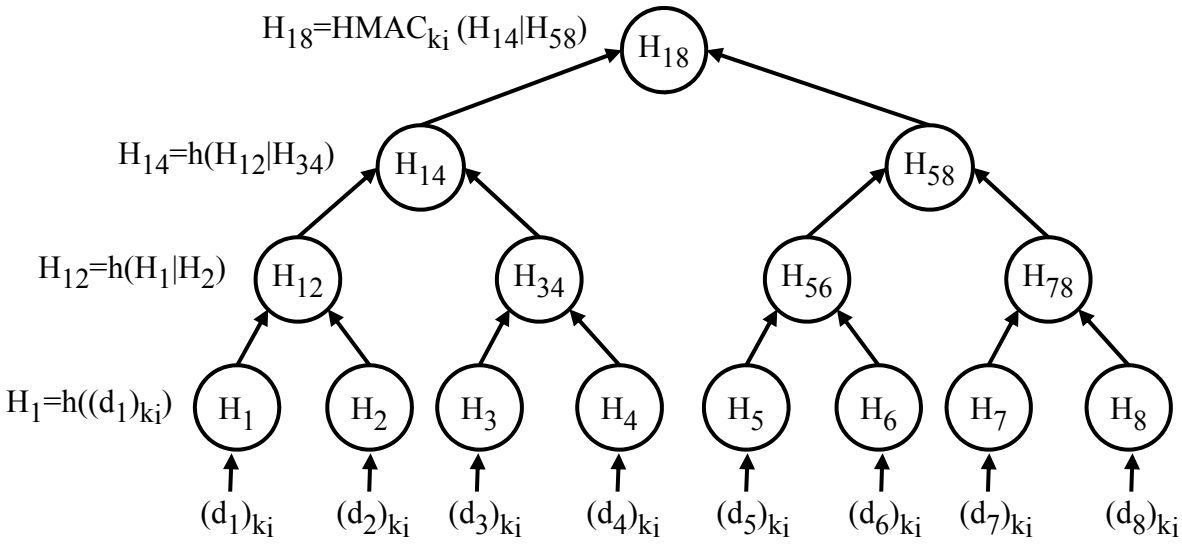


Figure 2.4. Merkle hash tree for 8 data items

The Merkle hash tree used in our solution has two special properties that allow the sink to verify query result integrity. First, the value of the root is computed using a keyed HMAC function, where the key is k_i, the key shared between sensor s_i and the sink. For example, in Figure 2.4, $H_{18} = HMAC_{k_i}(H_{14}|H_{58})$. Using a keyed HMAC function gives us the property that only sensor s_i and the sink can compute the root value. Second, the terminal nodes are arranged in an ascending order based on the value of each data item d_j.

We first discuss what a sensor needs to send to its nearest storage node along its data items. Each time sensor s_i wants to send n encrypted data items to a storage node, it first computes a Merkle hash tree over the n encrypted data items, and then sends the root value along with the n encrypted data items to a storage node. Note that among all the nodes in the Merkle hash tree, only the root is sent from sensor s_i to the storage node because the storage node can compute all other nodes in the Merkle hash tree by itself.

Next, we discuss what a storage node needs to send to the sink along a query result, *i.e.*, what should be included in a verification object. For the storage node that is near to sensor s_i, each time it receives a query $\{t, [a, b]\}$ from the sink, it first finds the data items that are in the range $[a, b]$. Second, it computes the Merkle hash tree (except the root) from the data items. Third, it sends the query result QR and the verification object VO to the sink. Given data items $(d_1)_{k_i}, \cdots, (d_n)_{k_i}$ in a storage node, where $d_1 < \cdots < d_n$, and a range $[a, b]$, where $d_{n_1-1} < a \leq d_{n_1} < \cdots < d_{n_2} \leq b < d_{n_2+1}$ and $1 \leq n_1 - 1 < n_2 + 1 \leq n$, and the query result $QR = \{(d_{n_1})_{k_i}, \cdots, (d_{n_2})_{k_i}\}$, the storage node should include $(d_{n_1-1})_{k_i}$ and $(d_{n_2+1})_{k_i}$ in the verification object VO because $(d_{n_1-1})_{k_i}$ and $(d_{n_2+1})_{k_i}$ ensure that the query result does include all data items that satisfy the query as the query result is bounded by them. We call $(d_{n_1-1})_{k_i}$ the *left bound* of the query result and $(d_{n_2+1})_{k_i}$ the *right bound* of the query result. Note that the left bound $(d_{n_1-1})_{k_i}$ and the right bound $(d_{n_2+1})_{k_i}$ may not exist. If $a \leq d_1$, the left bound $(d_{n_1-1})_{k_i}$ does not exist; if $d_n \leq b$, the right bound $(d_{n_2+1})_{k_i}$ does not exist. The verification object includes zero to two encrypted data items and $O(\log n)$

proof nodes in the Merkel hash tree that are needed for the sink to verify the integrity of the query result. Taking the example in Figure 2.5, suppose a storage node has received 8 data items $\{(2)_{k_i}, (5)_{k_i}, (9)_{k_i}, (15)_{k_i}, (20)_{k_i}, (23)_{k_i}, (34)_{k_i}, (40)_{k_i}\}$ that sensor s_i collected at time t, and the sink wants to perform the query $\{t, [10, 30]\}$ on the storage node. Using Theorem 1, the storage node finds that the query result includes $(15)_{k_i}$, $(20)_{k_i}$, and $(23)_{k_i}$ which satisfy the query. Along with the query result (*i.e.*, the three data items), the storage node also sends $(9)_{k_i}$, $(34)_{k_i}$, H_{12}, H_8, and H_{18}, which are marked grey in Figure 2.5, to the sink as the verification object.

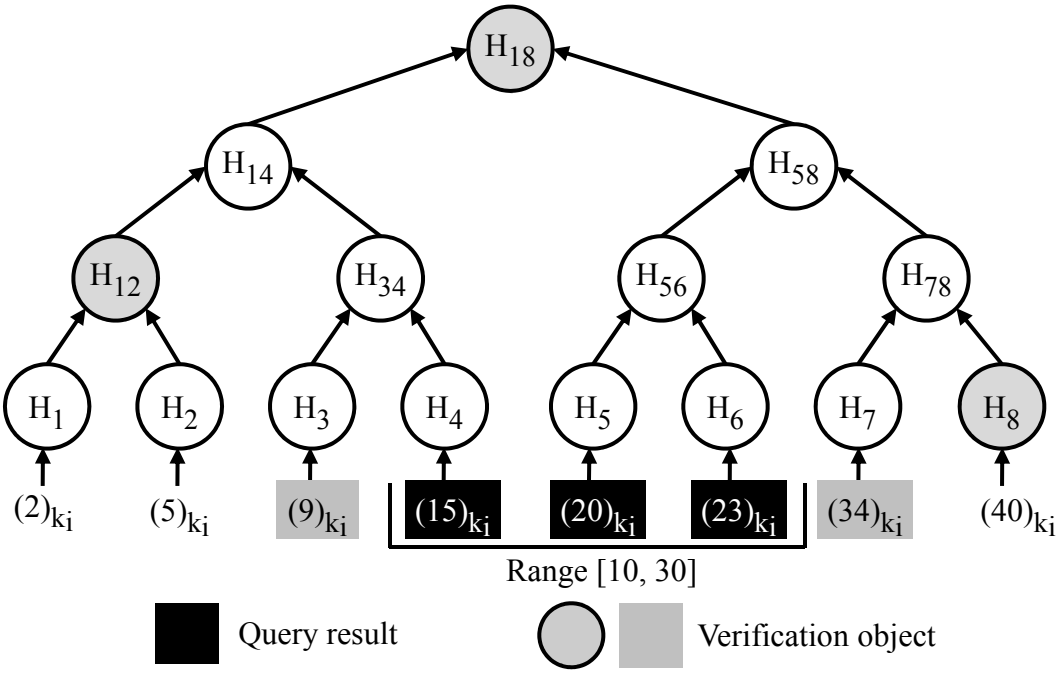


Figure 2.5. Data integrity verification

Next, we discuss how the sink uses Merkle hash trees to verify query result integrity. Upon receiving a query result $QR = \{(d_{n_1})_{k_i}, \cdots, (d_{n_2})_{k_i}\}$ and its verification object, the sink computes the root value of the Merkle hash tree and then verifies the integrity of the query result. Query result integrity is preserved if and only if the following four conditions hold: (1) The data items in the query result do satisfy the query. (2) If the left bound $(d_{n_1-1})_{k_i}$ exists,

verify that $d_{n_1-1} < a$ and $(d_{n_1-1})_{k_i}$ is the nearest left neighbor of $(d_{n_1})_{k_i}$ in the Merkle hash tree; otherwise, verify that $(d_{n_1})_{k_i}$ is the leftmost encrypted data item in the Merkle hash tree. (3) If the right bound $(d_{n_2+1})_{k_i}$ exists, verify that $b < d_{n_2+1}$ and $(d_{n_2+1})_{k_i}$ is the nearest right neighbor of $(d_{n_2})_{k_i}$ in the Merkle hash tree; otherwise, verify that $(d_{n_2})_{k_i}$ is the rightmost encrypted data item in the Merkle hash tree. (4) The computed root value is the same as the root value included in VO.

Note that sorting data items is critical in our scheme for ensuring the integrity of query result. Without this property, it is difficult for a storage node to prove query result integrity without sending all data items to the sink.

2.4.2 Integrity Scheme Using Neighborhood Chains

We first present a new data structure called *neighborhood chains* and then discuss its use in integrity verification. Given n data items $d_1, \cdots, d_n$, where $d_0 < d_1 < \cdots < d_n < d_{n+1}$, we call the list of n items encrypted using key k_i, $(d_0|d_1)_{k_i}, (d_1|d_2)_{k_i}, \cdots, (d_{n-1}|d_n)_{k_i}, (d_n|d_{n+1})_{k_i}$, the *neighborhood chain* for the n data items. Here "|" denotes concatenation. For any item $(d_{j-1}|d_j)_{k_i}$ in the chain, we call d_j the *value* of the item and $(d_j|d_{j+1})_{k_i}$ the *right neighbor* of the item. Figure 2.6 shows the neighborhood chain for the 5 data items 1, 3, 5, 7 and 9.

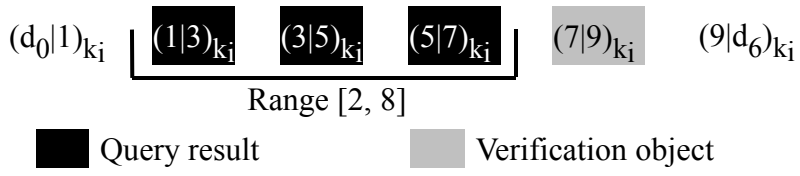


Figure 2.6. An example neighborhood chain

Preserving query result integrity using neighborhood chaining works as follows. After collecting n data items $d_1, \cdots, d_n$, sensor s_i sends the corresponding neighborhood chain $(d_0|d_1)_{k_i}$, $(d_1|d_2)_{k_i}$, $\cdots$, $(d_{n-1}|d_n)_{k_i}$, $(d_n|d_{n+1})_{k_i}$, instead of $(d_1)_{k_i}$, $\cdots$, $(d_n)_{k_i}$, to a stor-

age node. Given a range query $[a, b]$, the storage node computes QR as usual. The corresponding verification object VO only consists of the right neighbor of the largest data item in QR. Note that VO always consists of one item for any query. If $QR = \{(d_{n_1-1}|d_{n_1})_{k_i}, \cdots, (d_{n_2-1}|d_{n_2})_{k_i}\}$, then $VO = \{(d_{n_2}|d_{n_2+1})_{k_i}\}$; if $QR = \emptyset$, suppose $d_{n_2} < a \leq b < d_{n_2+1}$, then $VO = \{(d_{n_2}|d_{n_2+1})_{k_i}\}$.

After the sink receives QR and VO, it verifies the integrity of QR as follows. First, the sink verifies that every item in QR satisfies the query. Assume that the sink wants to perform the range query $[2, 8]$ over the data items in Figure 2.6. The storage node calculates QR to be $\{(1|3)_{k_i}, (3|5)_{k_i}, (5|7)_{k_i}\}$ and VO to be $\{(7|9)_{k_i}\}$. Second, the sink verifies that the storage node has not excluded any item that satisfies the query. Let $\{(d_{n_1-1}|d_{n_1})_{k_i}, \cdots, (d_{j-1}|d_j)_{k_i}, \cdots, (d_{n_2-1}|d_{n_2})_{k_i}\}$ be the correct query result and QR be the query result from the storage node. We consider the following four cases.

1. If there exists $n_1 < j < n_2$ such that $(d_{j-1}|d_j)_{k_i} \notin QR$, the sink can detect this error because the items in QR do not form a neighborhood chain.
2. If $(d_{n_1-1}|d_{n_1})_{k_i} \notin QR$, the sink can detect this error because it knows the existence of d_{n_1} from $(d_{n_1}|d_{n_1+1})_{k_i}$ and d_{n_1} satisfies the query.
3. If $(d_{n_2-1}|d_{n_2})_{k_i} \notin QR$, the sink can detect this error because it knows the existence of d_{n_2} from the item $(d_{n_2}|d_{n_2+1})_{k_i}$ in VO and d_{n_2} satisfies the query.
4. If $QR = \emptyset$, the sink can verify this fact because the item $(d_{n_2}|d_{n_2+1})_{k_i}$ in VO should satisfy the property $d_{n_2} < a \leq b < d_{n_2+1}$.

2.5 Queries over Multi-dimensional Data

Sensor collected data and sink issued queries are typically multi-dimensional as most sensors are equipped with multiple sensing modules such as temperature, humidity, pressure, etc.

A z-dimensional data item D is a z-tuple $(d^1, \cdots, d^z)$ where each d^l $(1 \leq l \leq z)$ is the value for the l-th dimension (*i.e.*, attribute). A z-dimensional range query consists of z sub-queries $[a^1, b^1], \cdots, [a^z, b^z]$ where each sub-query $[a^l, b^l]$ $(1 \leq l \leq z)$ is a range over the l-th dimension.

2.5.1 Privacy for Multi-dimensional Data

We extend our privacy preserving techniques for one-dimensional data to multi-dimensional data as follows. Let $D_1, \cdots, D_n$ denote the n z-dimensional data items that a sensor s_i collects at time slot t, where $D_j = (d_j^1, \cdots, d_j^z)$ $(1 \leq j \leq n)$. First, s_i encrypts these data with its secret key k_i. Second, for each dimension l, s_i applies the "magic" function $\mathcal{H}$ and obtains $\mathcal{H}(d_1^l, \cdots, d_n^l)$. At last, s_i sends the encrypted data items and $\mathcal{H}(d_1^1, \cdots, d_n^1)$, $\mathcal{H}(d_1^2, \cdots, d_n^2)$, $\cdots$, $\mathcal{H}(d_1^z, \cdots, d_n^z)$ to a nearby storage node. For example, sensor s_i collects 5 two-dimensional data items (1,11), (3,5), (6,8), (7,1) and (9,4) at time slot t, it will send the encrypted data items as well as $\mathcal{H}(1, 3, 6, 7, 9)$ and $\mathcal{H}(1, 4, 5, 8, 11)$ to a nearby storage node. When the sink wants to perform query $\{t, ([a^1, b^1], \cdots, [a^z, b^z])\}$ on a storage node, the sink applies the "magic" function $\mathcal{G}$ on each sub-query $[a^l, b^l]$ and sends $\{t, \mathcal{G}([a^1, b^1]), \cdots, \mathcal{G}([a^z, b^z])\}$ to the storage node. The storage node then applies the "magic" function $\mathcal{F}$ to find the query result QR^l for each sub-query $[a^l, b^l]$. Here the three "magic" functions $\mathcal{H}$, $\mathcal{G}$, and $\mathcal{F}$ are the same as the "magic" functions defined in Section 2.3. Finally, the storage node computes $QR = QR^1 \cap \cdots \cap QR^z$ as the query result. Considering the above example, given a range query ([2,7],[3,8]), the query result QR^1 for the sub-query [2,6] is the encrypted data items of (3,5),(6,8) and the query result QR^2 for the sub-query [3,8] is the encrypted data items of (9,4),(3,5),(6,8). Therefore the query result QR is the encrypted data items of (3,5),(6,8).

2.5.2 Integrity for Multi-dimensional Data

We next present two integrity preserving schemes for multi-dimensional data: one builds a merkle hash tree for each dimension; another builds a multi-dimensional neighborhood chain.

Integrity Scheme Using Merkle Hash Trees

To preserve integrity, sensor s_i first computes z Merkle hash trees over the n encrypted data items along z dimensions. In the l-th Merkle hash tree, the n data items are sorted according to the values of the l-th attribute. Second, s_i sends the z root values to a storage node along with the encrypted data items. For a storage node that is near to sensor s_i, each time it receives a query $\{t, ([a^1, b^1], \cdots, [a^z, b^z])\}$, it first finds the query result QR^l for each range $[a^l, b^l]$. Second, it chooses the query result $QR^{l'}$ that contains the smallest number of encrypted data items among $QR^1, \cdots, QR^n$. Third, it computes the Merkle hash tree in which the data items are sorted according to the l'-th attribute. Finally, it sends $QR^{l'}$ and the corresponding verification object $VO^{l'}$ to the sink. For example, suppose a sensor s_i collects 4 two-dimensional data items $\{(12, 5), (15, 6), (23, 4), (45, 3)\}$ in a time slot. Sensor s_i computes a Merkle hash tree along each dimension. Figure 2.7 shows the two Merkle hash trees. Given a two-dimensional range query $\{[16, 23], [3, 5]\}$, the storage node can find the query results $QR^1 = \{(23, 4)_{k_i}\}$ based on the first attribute and $QR^2 = \{(45, 3)_{k_i}, (23, 4)_{k_i}, (12, 5)_{k_i}\}$ based on the second attribute. Since QR^1 only contains one encrypted data item, the storage node sends to the sink the query result $QR^1 = \{(23, 4)_{k_i}\}$ and the corresponding verification object $VO^1 = \{(15, 6)_{k_i}, (45, 3)_{k_i}, H_1^1, H_{14}^1\}$.

Note that the query result of a multi-dimensional range query may contain data items that do not satisfy the query. After decryption, the sink can easily prune the query result by discarding such data items.

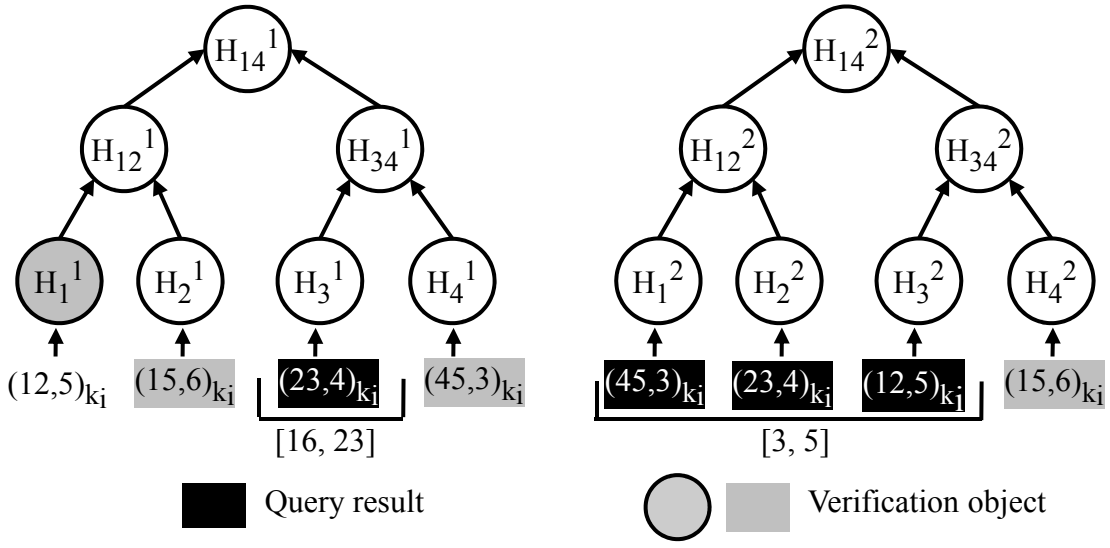


Figure 2.7. Merkle hash trees for two-dimensional data

Integrity Scheme Using Neighborhood Chains

The basic idea is for each of the z values in a data item, we find its nearest left neighbor along each dimension and embed this information when we encrypt the item. Such neighborhood information is used by the sink for integrity verification.

We first present *multi-dimensional neighborhood chains* and then discuss its use in integrity verification. Let $D_1, \cdots, D_n$, where $D_j = (d_j^1, \cdots, d_j^z)$ for each $1 \leq j \leq n$, denote n z-dimensional data items. We use d_0^l and d_{n+1}^l to denote the lower bound and the upper bound of any data item along dimension l. We call $D_0 = (d_0^1, \cdots, d_0^z)$ and $D_{n+1} = (d_{n+1}^1, \cdots, d_{n+1}^z)$ the lower bound and upper bound of the data items. For each dimension $1 \leq l \leq z$, we can sort the values of n data items along the l-th dimension together with d_0^l and d_{n+1}^l in an ascending order. For ease of presentation, we assume $d_0^l < d_1^l < \cdots < d_{n+1}^l$ for every dimension $1 \leq l \leq z$. In this sorted list, we call d_{j-1}^l $(1 \leq j \leq n+1)$ the *left neighboring value* of d_j^l. We use $\mathcal{L}^l(d_j^l)$ to denote the left neighboring value of d_j^l along dimension l. A *multi-dimensional neighborhood chain* for $D_1, \cdots, D_n$ is constructed by encrypting every item D_j as $(\mathcal{L}^1(d_j^1)|d_j^1, \cdots, \mathcal{L}^z(d_j^z)|d_j^z)_{k_i}$, which is denoted as $MNC(D_j)$.

We call D_j the value of $MNC(D_j)$. Note that when multiple data items have the same value along the l-th dimension, we annotate $\mathcal{L}^l(d_j^l)$ with the number of such items in $MNC(D_j)$. The list of $n+1$ items encrypted with key k_i, $MNC(D_1)$, $\cdots$, $MNC(D_n)$, $MNC(D_{n+1})$, forms a *multi-dimensional neighborhood chain*. The nice property of a multi-dimensional neighborhood chain is that all data items form a neighborhood chain along every dimension. This property allows the sink to verify the integrity of query results. Considering 5 example 2-dimensional data items (1,11), (3,5), (6,8), (7,1), (9,4) with lower bound $(0, 0)$ and upper bound $(15, 15)$, the corresponding multi-dimensional neighborhood chain encrypted with key k_i is $(0|1, 9|11)_{k_i}$, $(1|3, 4|5)_{k_i}$, $(3|6, 5|8)_{k_i}$, $(6|7, 0|1)_{k_i}$, $(7|9, 1|4)_{k_i}$ and $(9|15, 11|15)_{k_i}$. Figure 2.8 illustrates this chain, where each black point denotes an item, the two grey points denote the lower and upper bounds, the solid arrows illustrate the chain along the X dimension, and the dashed arrows illustrate the chain along the Y dimension.

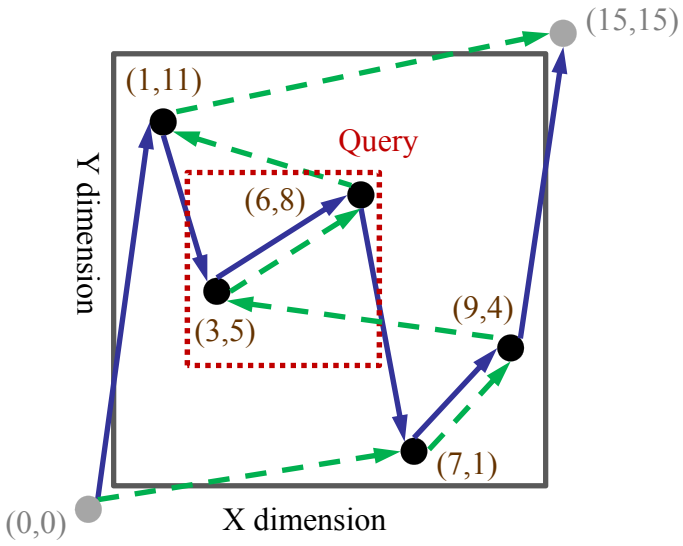


Figure 2.8. A 2-dimensional neighborhood chain

Next, we discuss the operations carried on sensors, storage nodes, and the sink using multi-dimensional chaining.

Sensors: After collecting n z-dimensional data items at time slot t, sensor s_i computes

the multi-dimensional chain for the items and sends it to a storage node.

Storage nodes: Given a z-dimensional query $([a^1, b^1], \cdots, [a^z, b^z])$, a storage node first computes $QR = QR^1 \cap \cdots \cap QR^z$. Second, it computes $VO = (QR^l - QR) \cup \{\mathcal{R}([a^l, b^l])\}$, where QR^l is the smallest set among $QR^1, \cdots, QR^z$ (*i.e.*, $|QR^l| \leq |QR^i|$ for any $1 \leq i \leq z$) and $\mathcal{R}([a^l, b^l])$ is the right bounding item of the range $[a^l, b^l]$. Given a multi-dimensional chain $MNC(D_1)$, $\cdots$, $MNC(D_n)$, $MNC(D_{n+1})$ and a sub-query $[a^l, b^l]$ along dimension l, the *right bounding item* of $[a^l, b^l]$ is the item $MNC(D_j)$ where $\mathcal{L}^l(d_j^l) \leq b^l < d_j^l$. Figure 2.8 shows a query ([2,6],[3,8]) with a query result $QR = \{MNC(3,5),\ MNC(6,8)\}$ and $VO = \{MNC(7,1)\}$.

The Sink: Upon receiving QR and VO, the sink verifies the integrity of QR as follows. First, it verifies that every item in QR satisfies the query. Second, it verifies that the storage node has not excluded any item that satisfies the query based on the following three properties: (1) The items in $QR \cup VO$ should form a chain along one dimension, say l. Thus, if the storage node excludes an item whose value in the l-th dimension is in the middle of this chain, this chaining property would be violated. (2) The item in $QR \cup VO$ that has the smallest value among the l-th dimension, say $MNC(D_j)$, satisfies the condition that $\mathcal{L}^l(d_j^l) < a^l$. Thus, if the storage node excludes the item whose value on the l-th dimension is the beginning of the chain, this property would be violated. (3) There exists only one item in VO that is the right bounding item of $[a^l, b^l]$. Thus, if the storage node excludes the item whose value on the l-th dimension is the end of the chain, this property would be violated.

2.6 SafeQ Optimization

In this section, we present an optimization technique based on Bloom filters [15] to reduce the communication cost between sensors and storage nodes. This cost can be significant because of two reasons. First, in each submission, a sensor needs to convert each range

$[d_j, d_{j+1}]$, where d_j and d_{j+1} are two numbers of w bits, to $2w-2$ prefix numbers in the worst case. Second, the sensor applies HMAC to each prefix number, which results in a 128-bit string if we choose HMAC-MD5 or a 160-bit string if we choose HMAC-SHA1. Reducing communication cost for sensors is important because of power consumption reasons.

Our basic idea is to use a Bloom filter to represent $HMAC_g(\mathcal{N}(\mathcal{S}([d_0, d_1])))$, $\cdots$, $HMAC_g(\mathcal{N}(\mathcal{S}([d_n, d_{n+1}])))$. Thus, a sensor only needs to send the Bloom filter instead of the hashes to a storage node. The number of bits needed to represent the Bloom filter is much smaller than that needed to represent the hashes. Next, we discuss the operations that sensors and storage nodes need to perform in using this optimization technique.

Sensors: Let A be a bit array of size c representing the Bloom filter for $HMAC_g(\mathcal{N}(\mathcal{S}([d_0, d_1])))$, $\cdots$, $HMAC_g(\mathcal{N}(\mathcal{S}([d_n, d_{n+1}])))$ that a sensor computes after collecting n data items $d_1, \cdots, d_n$ assuming $d_1 \leq \cdots \leq d_n$. Let B be an array of c pointers. For every $0 \leq j \leq n$ and for every number v in $HMAC_g(\mathcal{N}(\mathcal{S}([d_j, d_{j+1}])))$, the sensor applies k hash functions on v, where each hash function h_i $(1 \leq i \leq k)$ hashes v to an integer in the range $[1, c]$, and then sets $A[h_i(v)]$ to be 1 and appends the index j to the list that $B[h_i(v)]$ points to. In each submission, the sensor sends A and B to its closest storage node. For example, $HMAC_g(\mathcal{N}(\mathcal{S}([10, 11])))$ and $HMAC_g(\mathcal{N}(\mathcal{S}([11, 15])))$ can be represented as the two arrays in Figure 2.9, where "-" denotes an empty pointer. Note that $\mathcal{N}(\mathcal{S}([10, 11])) = \{10110\}$ and $\mathcal{N}(\mathcal{S}([11, 15])) = \{10111, 11100\}$.

The logical meaning of B is an array of pointers, each pointing to a list of indices from 0 to n. To reduce the space used for storing pointers, we implement B as a concatenation of all these lists separated by delimiters. For example, we can represent the array B in Figure 2.9 as a list $\langle 1, \backslash 0, 1, \backslash 0, 1, 2, \backslash 0, 2, \backslash 0, 2, \backslash 0, 2, \backslash 0, 2, \backslash 0, \cdots, \backslash 0, 2\rangle$.

Storage nodes: Recall that $a \in [d_j, d_{j+1}]$ if and only if $HMAC_g(\mathcal{N}(\mathcal{F}(a))) \cap HMAC_g(\mathcal{N}(\mathcal{S}([d_j, d_{j+1}]))) \neq \emptyset$. If $HMAC_g(\mathcal{N}(\mathcal{F}(a))) \cap HMAC_g(\mathcal{N}(\mathcal{S}([d_j, d_{j+1}]))) \neq \emptyset$, then there exists at least one number v in $HMAC_g(\mathcal{N}(\mathcal{F}(a)))$ such that the following two

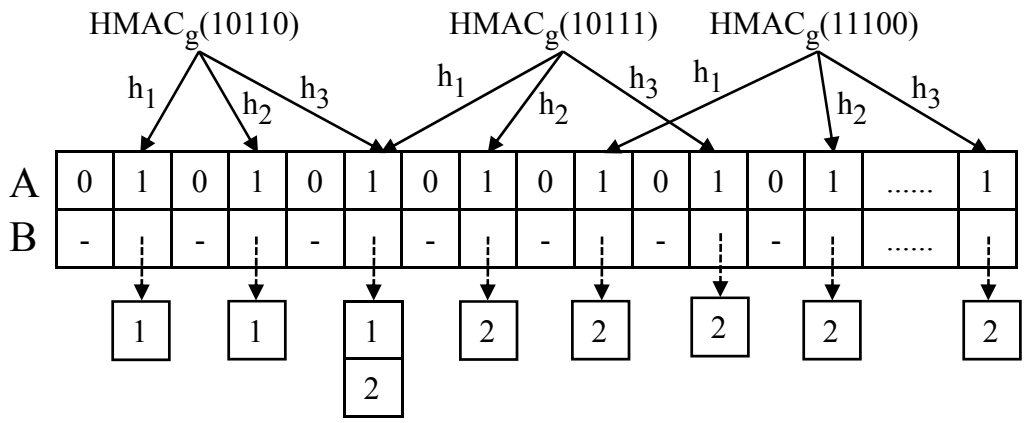


Figure 2.9. An example Bloom filter

conditions hold: (1) for every $1 \leq i \leq k$, $A[h_i(v)]$ is 1; (2) for every $1 \leq i \leq k$, index j is included in the list that $B[h_i(v)]$ points to. For example, to verify whether $12 \in [11, 15]$ using the Bloom filter in Figure 2.9, a storage node can apply the 3 hash functions to each number in $HMAC_g(\mathcal{N}(\mathcal{F}(12)))$. For one number $HMAC_g(11100)$ in $HMAC_g(\mathcal{N}(\mathcal{F}(12)))$, the storage node verifies that $HMAC_g(11100)$ satisfies the above two conditions, therefore $12 \in [11, 15]$.

Although using our optimization technique QR may contain data items that do not satisfy the query, they can be easily pruned by the sink after decryption. Given a query $[a, b]$, using this optimization technique, a storage node may find multiple ranges that contain a and multiple ranges that contain b due to false positives of Bloom filters. In this case, the storage node uses the first range that contains a and the last range that contains b to compute the query result.

Bloom filters introduce false positives in the result of a query, *i.e.*, the data items that do not satisfy the query. We can control the false positive rate by adjusting Bloom filter parameters. Let ϵ denote the average false positive rate and w_h denote the bit length of each number in $HMAC_g(\mathcal{N}(\mathcal{S}([d_j, d_{j+1}])))$. For simplicity, we assume that each set $HMAC_g(\mathcal{N}(\mathcal{S}([d_j, d_{j+1}])))$ contains the same number of values, which is denoted as q. The

upper bound of the average false positive rate ϵ is shown in Formula 2.1, the derivation of which is in Appendix A.

$$\epsilon < \frac{1}{3}\frac{(n+2)(n+3)}{(n+1)^{k-1}}(1-e^{-k(n+1)q/c})^k \tag{2.1}$$

To represent $HMAC_g(\mathcal{N}(\mathcal{S}([d_0,d_1]))),\ \cdots,\ HMAC_g(\mathcal{N}(\mathcal{S}([d_n,d_{n+1}])))$, without Bloom filters, the total number of bits required is $w_h(n+1)q$; with Bloom filters, the total number of bits required is at most $c+2k(n+1)q\lceil\log_2(n+1)\rceil$, the calculation of which is in Appendix A. Therefore, our optimization technique reduces the communication cost if

$$w_h(n+1)q > c + 2k(n+1)q\lceil\log_2(n+1)\rceil \tag{2.2}$$

Based on Formula 2.1 and 2.2, assuming $w_h = 128$ and $n \geq 3$, to achieve reduction on the communication cost of sensors and the small false positive rate of $\epsilon < 1\%$, we choose $c = \frac{1}{\ln 2}k(n+1)q$ and k to be $4 \leq k < \frac{128}{1.44+2\lceil\log_2(n+1)\rceil}$. Note that only when $n \geq 2^{15}$, which is unlikely to happen, such k does not exist.

2.7 Queries in Event-driven Networks

So far we have assumed that at each time slot, a sensor sends to a storage node the data that it collected at that time slot. However, this assumption does not hold for event-driven networks, where a sensor only reports data to a storage node when certain event happens. If we directly apply our solution here, then the sink cannot verify whether a sensor collected data at a time slot. The case that a sensor did not submit any data at time slot t and the case that the storage node discards all the data that the sensor collected at time slot t are not distinguishable for the sink.

We address the above challenge by sensors reporting their idle period to storage node each time when they submit data after an idle period or when the idle period is longer than a threshold. Storage nodes can use such idle period reported by sensors to prove to the sink that a sensor did not submit any data at any time slot in that idle period. Next, we discuss the operations carried on sensors, storage nodes and the sink.

Sensors: An *idle period* for a sensor is a time slot interval $[t_1, t_2]$, which indicates that the sensor has no data to submit from t_1 to t_2, including t_1 and t_2. Let γ be the threshold of a sensor being idle without reporting to a storage node. Suppose the last time that sensor s_i submitted data or reported idle period is time slot $t_1 - 1$. At any time slot $t \geq t_1$, s_i acts based on three cases:

1. $t = t_1$: In this case, if s_i has data to submit, then it just submits the data; otherwise it takes no action.
2. $t_1 < t < \gamma + t_1 - 1$: In this case, if s_i has data to submit, then it submits data along with encrypted idle period $[t_1, t-1]_{k_i}$; otherwise it takes no action. We call $[t_1, t-1]_{k_i}$ an *idle proof*.
3. $t = \gamma + t_1 - 1$: In this case, if s_i has data to submit, then it submits data along with the idle proof $[t_1, t-1]_{k_i}$; otherwise, it submits the idle proof $[t_1, t]_{k_i}$.

Figure 2.10 illustrates some idle periods for sensor s_i, where each unit in the time axis is a time slot, a grey unit denotes that s_i has data to submit at that time slot, and a blank unit denotes that s_i has no data to submit at that time slot. According to the second case, at time slot $t_2 + 1$, s_i submits data along with the idle proof $[t_1, t_2]_{k_i}$. According to the third case, at time slot t_4, s_i submits the idle proof $[t_3, t_4]_{k_i}$.

Storage nodes: When a storage node receives a query $\{t, \mathcal{G}([a, b])\}$ from the sink, it first checks whether s_i has submitted data at time slot t. If s_i has, then the storage node sends the query result as discussed in Section 2.3. Otherwise, the storage node checks whether s_i

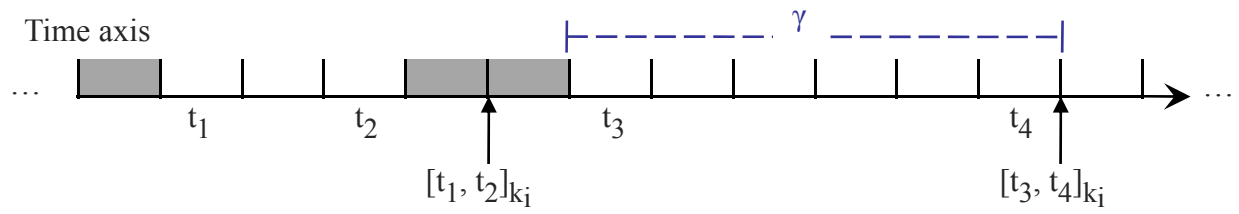


Figure 2.10. Example idle periods and data submissions

has submitted an idle proof for an idle period containing time slot t. If true, then it sends the idle proof to the sink as VO. Otherwise, it replies to the sink saying that it does not have the idle proof containing time slot t at this moment, but once the right idle proof is received, it will forward to the sink. The maximum number of time slots that the sink may need to wait for the right idle proof is $\gamma - 1$. Here γ is a system parameter trading off efficiency and the amount of time that sink may have to wait for verifying data integrity. Smaller γ favors the sink for integrity verification and larger γ favors sensors for power saving because of less communication cost.

The Sink: Changes on the sink side are minimal. In the case that VO lacks the idle proof for verifying the integrity of QR, it will defer the verification for at most $\gamma - 1$ time slots, during which benign storage nodes are guaranteed to send the needed idle proof.

2.8 Complexity and Security Analysis

2.8.1 Complexity Analysis

Assume that a sensor collects n z-dimensional data items in a time slot, each attribute of a data item is a w_o-bit number, and the HMAC result of each numericalized prefix is a w_h number. The computation cost, communication cost, and storage space of SafeQ are described in the following table. Note that the communication cost denotes the number of

bytes sent for each submission or query, and the storage space denotes the number of bytes stored in a storage node for each submission.

	Computation	Communication	Space
Sensor	$O(w_o zn)$ hash $O(n)$ encryption	$O(w_o w_h zn)$	–
Storage node	$O(w_o z)$ hash	$O(zn)$	$O(w_o w_h zn)$
Sink	$O(w_o z)$ hash	$O(w_o z)$	–

Table 2.2. Complexity analysis of SafeQ

2.8.2 Privacy Analysis

In a SafeQ protected two-tiered sensor network, compromising a storage node does not allow the attacker to obtain the actual values of sensor collected data and sink issued queries. The correctness of this claim is based on the fact that the hash functions and encryption algorithms used in SafeQ are secure. In the submission protocol, a storage node only receives encrypted data items and the secure hash values of prefixes converted from the data items. Without knowing the keys used in the encryption and secure hashing, it is computationally infeasible to compute the actual values of sensor collected data and the corresponding prefixes. In the query protocol, a storage node only receives the secure hash values of prefixes converted from a range query. Without knowing the key used in the secure hashing, it is computationally infeasible to compute the actual values of sink issued queries.

Next, we analyze information leaking if $HMAC_g()$ does not satisfy the one-wayness property. More formally, given y, where $y = HMAC_g(x)$ and x is a numericalized prefix, suppose that a storage node takes $O(T)$ steps to compute x. Recall that the number of HMAC hashes sent from a sensor is $O(w_o zn)$. To reveal a data item d_j, the storage node needs to reveal all the numericalized prefixes in $HMAC_g(\mathcal{N}(\mathcal{S}([d_{j-1}, d_j])))$. Thus, to reveal n data items, the storage node would take $O(w_o znT)$ steps. Here $T = 2^{128}$ for HMAC.

2.8.3 Integrity Analysis

For our scheme using Merkle hash trees, the correctness of this claim is based on the property that any change of leaf nodes in a Merkle hash tree will change the root value. Recall that the leaf nodes in a Merkle hash tree are sorted according to their values. In a query response, the left bound of the query result (if it exists), the query result, and the right bound of the query result (if it exists) must be consecutive leaf nodes in the Merkle hash tree. If the storage node includes forged data in the query result or excludes a legitimate data item from the query result, the root value computed at the sink will be different from the root value computed at the corresponding sensor.

For our scheme using neighborhood chains, the correctness is based on the following three properties that QR and VO should satisfy for a query. First, items in $QR \cup VO$ form a chain. Excluding any item in the middle or changing any item violates the chaining property. Second, the first item in $QR \cup VO$ contains the value of its left neighbor, which should be out of the range query on the smaller end. Third, the last item in $QR \cup VO$ contains the value of its right neighbor, which should be out of the range query on the larger end.

2.9 Experimental Results

2.9.1 Evaluation Methodology

To compare SafeQ with the state-of-the-art, which is represented by S&L scheme, we implemented both schemes and performed side-by-side comparison on a large real data set. We measured average power and space consumption for both the submission and query protocols of both schemes.

2.9.2 Evaluation Setup

We implemented both SafeQ and S&L scheme using TOSSIM [8], a widely used wireless sensor network simulator. We measured the efficiency of SafeQ and S&L scheme on 1, 2, and 3 dimensional data. For better comparison, we conducted our experiments on the same data set that S&L used in their experiment [69]. The data set was chosen from a large real data set from Intel Lab [4] and it consists of the temperature, humidity, and voltage data collected by 44 nodes during 03/01/2004-03/10/2004. Each data attribute follows Gaussian distribution. Note that S&L only conducted experiments on the temperature data, while we experimented with both SafeQ and S&L schemes on 1-dimensional data (of temperature), 2-dimensional data (of temperature and humidity) and 3-dimensional data (of temperature, humidity, and voltage).

In implementing SafeQ, we used HMAC-MD5 [46] with 128-bit keys as the hash function for hashing prefix numbers. We used the DES encryption algorithm in implementing both SafeQ and S&L scheme. In implementing our Bloom filter optimization technique, we chose the number of hash functions to be 4 (*i.e.*, $k = 4$), which guarantees that the false positive rate induced by the Bloom filter is less than 1%. In implementing S&L scheme, we used the parameter values (*i.e.*, $VAR_p = 0.4$ and $EN_p = 1$), which are corresponding to the minimum false positives of query results in their experiments, for computing optimal bucket partitions as in [69], and we used HMAC-MD5 with 128-bit keys as the hash function for computing encoding number. For multi-dimensional data, we used their optimal bucket partition algorithm to partition multi-dimensional data along each dimension. In our experiments, we experimented with different sizes of time slots ranging from 10 minutes to 80 minutes. For each time slot, we generated 1,000 random range queries in the form of $([a^1, b^1], [a^2, b^2], [a^3, b^3])$, where a^1, b^1 are two random values of temperature, a^2, b^2 are two random values of humidity, and a^3, b^3 are two random values of voltage.

2.9.3 Evaluation Results

The experimental results from our side-by-side comparison show that SafeQ significantly outperforms S&L scheme for multi-dimensional data in terms of power and space consumption. For the two integrity preserving schemes, the neighborhood chaining technique is better than Merkle hash tree technique in terms of both power and space consumption. The rationale for us to include the Merkle hash tree based scheme is that Merkle hash trees are the typical approach to achieving integrity. We use *SafeQ-MHT+* and *SafeQ-MHT* to denote our schemes using Merkle hash trees with and without Bloom filters, respectively, and we use *SafeQ-NC+* and *SafeQ-NC* to denote our schemes using neighborhood chains with and without Bloom filters, respectively.

Figures 2.11(a), 2.11(b), and 2.12(a), show the average power consumption of sensors for 3-dimensional, 2-dimensional, and 1-dimensional data, respectively, versus different sizes of time slots. Figures 2.13(a), 2.13(b), and 2.14(a), show the average power consumption of storage nodes for 3-dimensional, 2-dimensional, and 1-dimensional data, respectively, versus different sizes of time slots. We observe that the power consumption of both sensors and storage nodes grows linearly with the number of data items, which confirms our complexity analysis in Section 4.5.2. Note that the number of collected data items is in direct proportion to the size of time slots. For power consumption, in comparison with S&L scheme, our experimental results show that for 3-dimensional data, SafeQ-NC+ consumes 184.9 times less power for sensors and 76.8 times less power for storage nodes; SafeQ-MHT+ consumes 171.4 times less power for sensors and 46.9 times less power for storage nodes; SafeQ-NC consumes 59.2 times less power for sensors and 76.8 times less power for storage nodes; SafeQ-MHT consumes 57.9 times less power for sensors and 46.9 times less power for storage nodes. For 2-dimensional data, SafeQ-NC+ consumes 10.3 times less power for sensors and 9.0 times less power for storage nodes; SafeQ-MHT+ consumes 9.5 times less power for

sensors and 5.4 times less power for storage nodes; SafeQ-NC consumes 2.7 times less power for sensors and 9.0 times less power for storage nodes; SafeQ-MHT consumes 2.6 times less power for sensors and 5.4 times less power for storage nodes. Our experimental results conform with the theoretical analysis that the power consumption in S&L scheme grows exponentially with the number of dimensions, whereas in SafeQ it grows linearly with the number of dimensions times the number of data items.

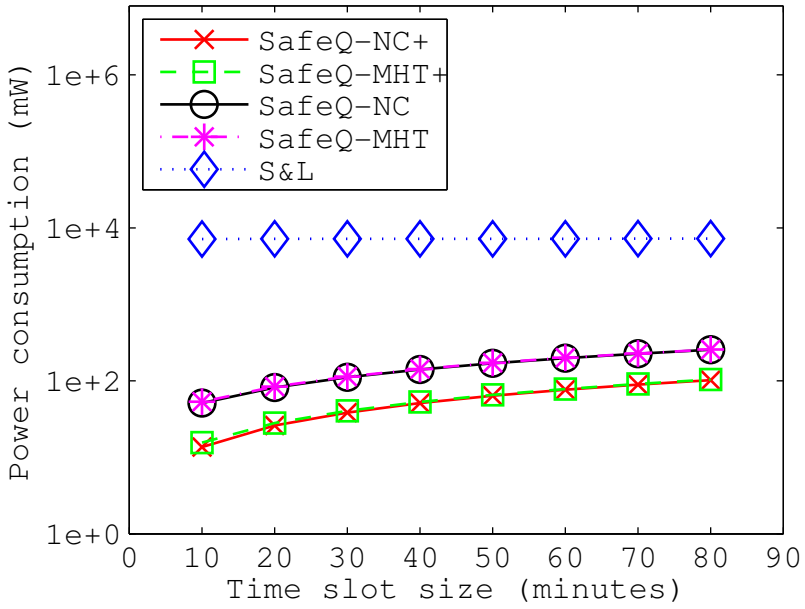
Figures 2.12(b) and 2.14(b) show the average power consumption for a 10-minute slot for a sensor and a storage node, respectively, versus the number of dimensions of the data. We observe that there are almost linear correlation between the average power consumption for both sensors and storage nodes and the number of dimensions of the data, which also confirms our complexity analysis in Section 4.5.2.

Our experimental results also show that SafeQ is comparable to S&L scheme for 1-dimensional data in terms of power and space consumption. For power consumption, SafeQ-NC+ consumes about the same power for sensors and 0.7 times less power for storage nodes; SafeQ-MHT+ consumes about the same power for sensors and 0.3 times less power for storage nodes; SafeQ-NC consumes 1.0 times more power for sensors and 0.7 times less power for storage nodes; SafeQ-MHT consumes 1.0 times more power for sensors and 0.3 times less power for storage nodes. For space consumption on storage nodes, SafeQ-NC+ and SafeQ-MHT+ consume about the same space, and SafeQ-NC and SafeQ-MHT consume about 1.0 times more space.

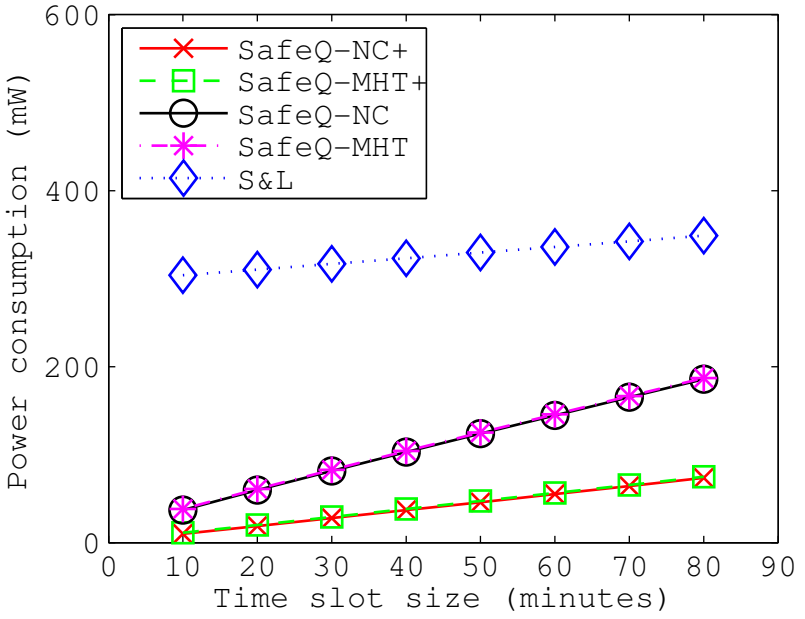
Figures 2.15(a), 2.15(b) and 2.16(a) show the average space consumption of storage nodes for 3, 2 and 1 dimensional data, respectively. For space consumption on storage nodes, in comparison with S&L scheme, our experimental results show that for 3-dimensional data, SafeQ-NC+ consumes 182.4 times less space; SafeQ-MHT+ consumes 169.1 times less space; SafeQ-NC consumes 58.5 times less space; SafeQ-MHT consumes 57.2 times less space. For 2-dimensional data, SafeQ-NC+ consumes 10.2 times less space; SafeQ-MHT+ consumes 9.4

times less space; SafeQ-NC consumes 2.7 times less space; SafeQ-MHT consumes 2.6 times less space. The results conform with the theoretical analysis that the space consumption in S&L scheme grows exponentially with the number of dimensions, whereas in SafeQ it grows linearly with the number of dimensions and the number of data items.

Figure 2.16(b) shows the average space consumption of storage nodes for each data item versus the number of dimensions of the data item. For each 3-dimensional data item, S&L consumes about over 10^4 bytes, while SafeQ-NC+ and SafeQ-MHT+ consume only 40 bytes.

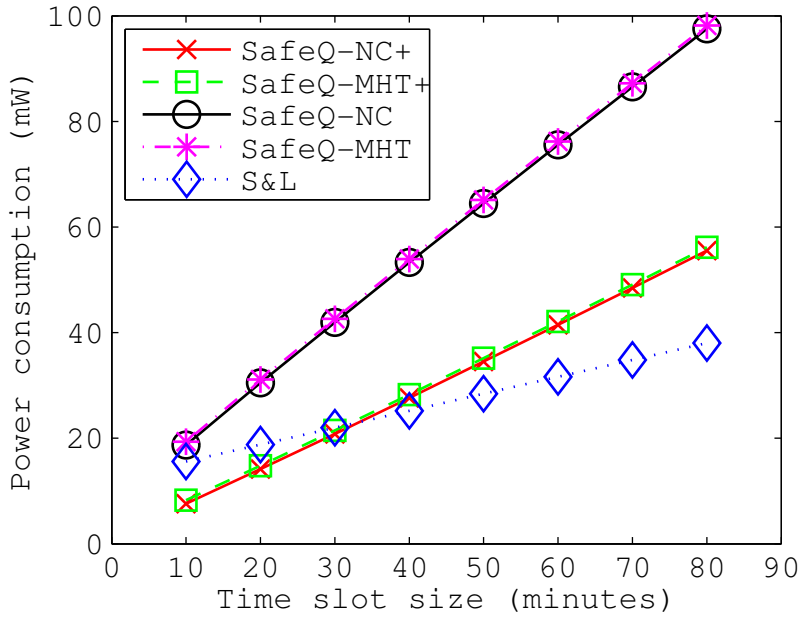


(a) 3-dimensional data

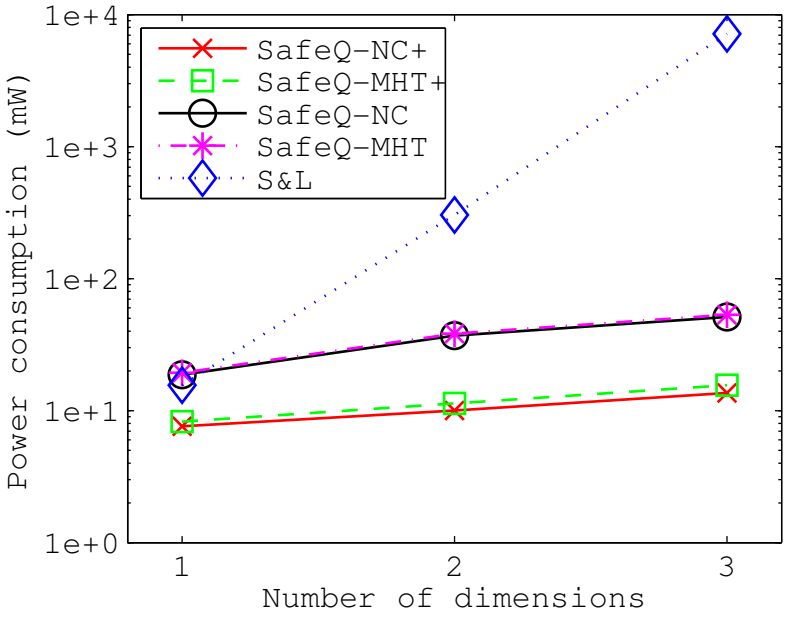


(b) 2-dimensional data

Figure 2.11. Average power consumption per submission for a sensor (A)

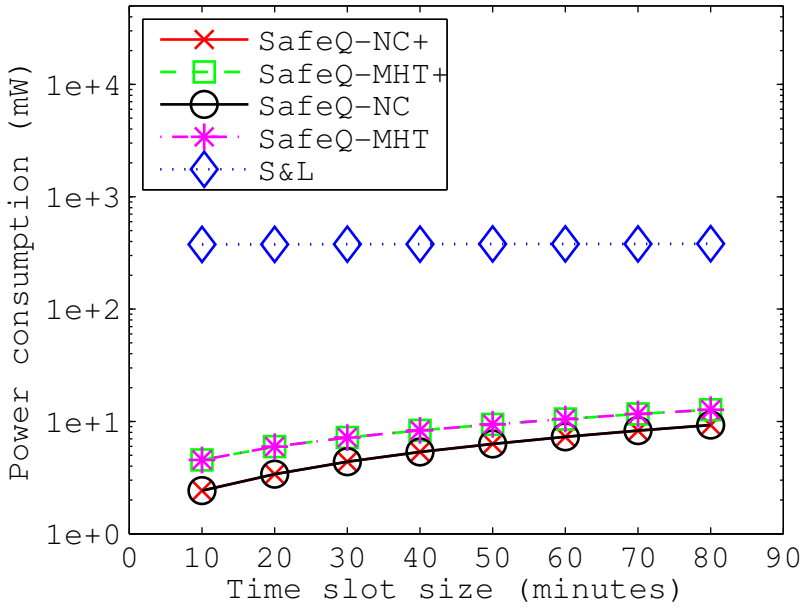


(a) 1-dimensional data

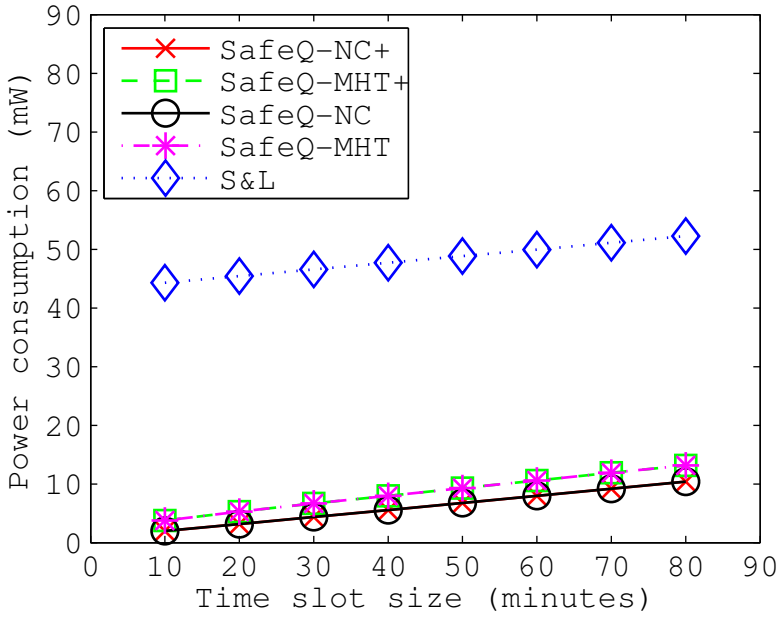


(b) for 10 minutes

Figure 2.12. Average power consumption per submission for a sensor (B)

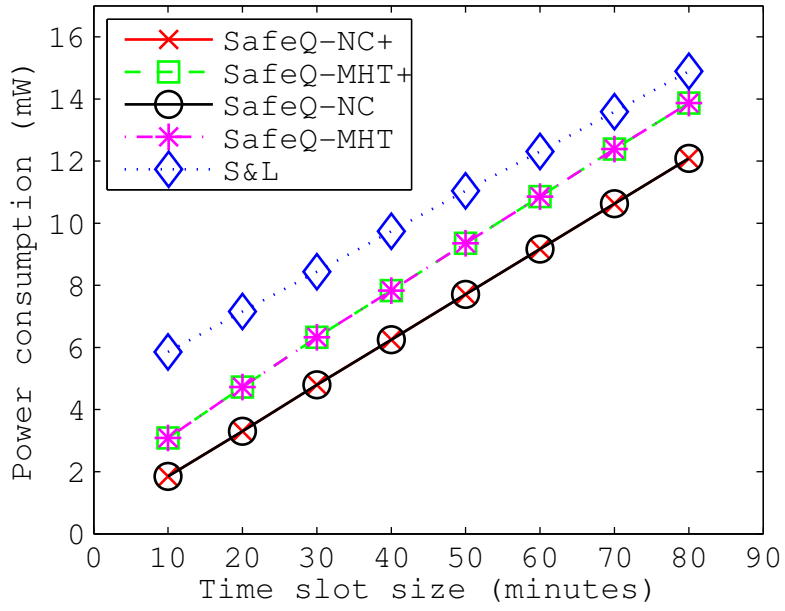


(a) 3-dimensional data

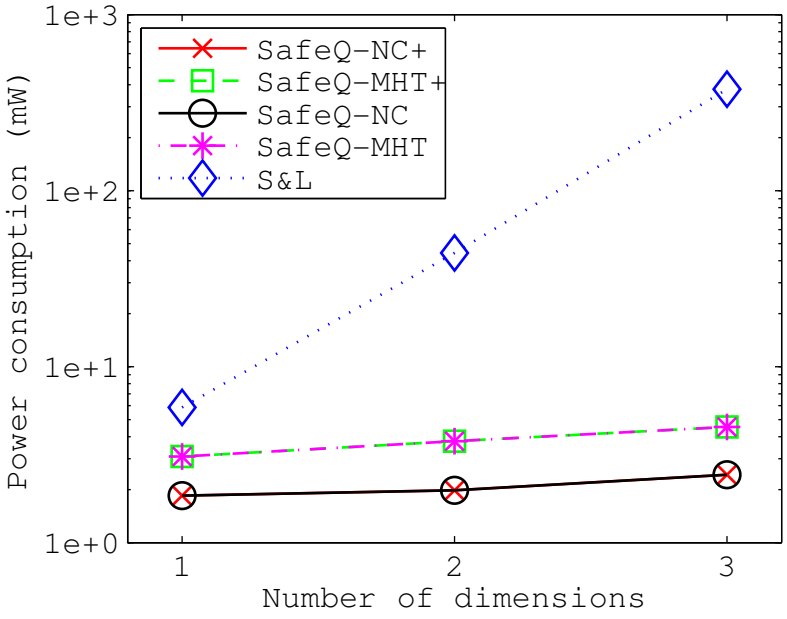


(b) 2-dimensional data

Figure 2.13. Average power consumption per query response for a storage node (A)

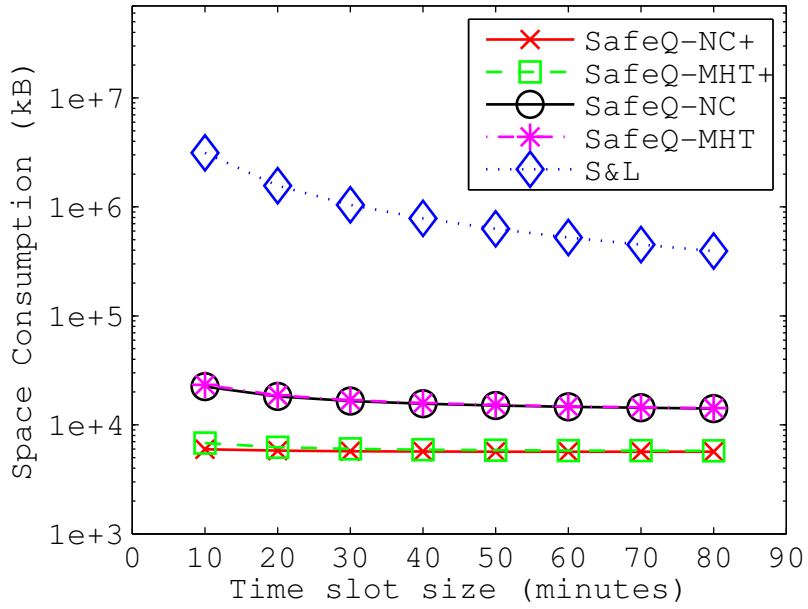


(a) 1-dimensional data

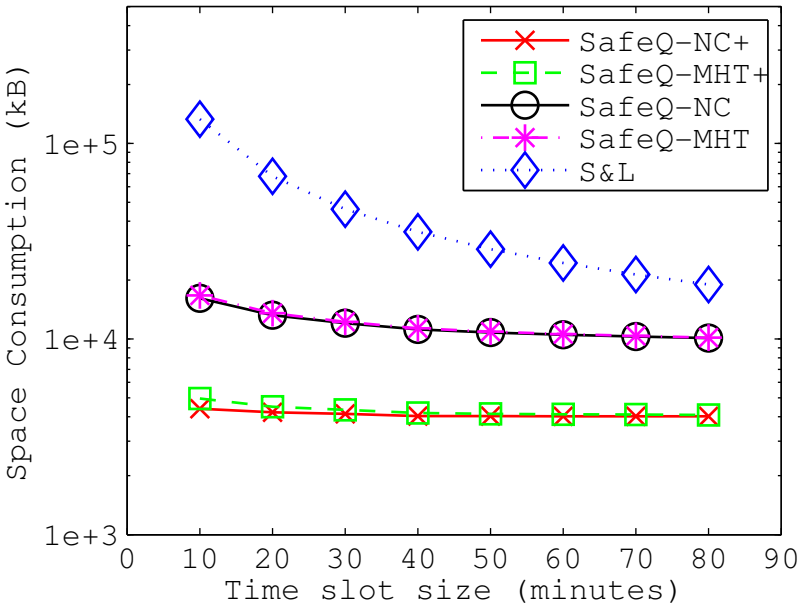


(b) for 10 minutes

Figure 2.14. Average power consumption per query response for a storage node (B)

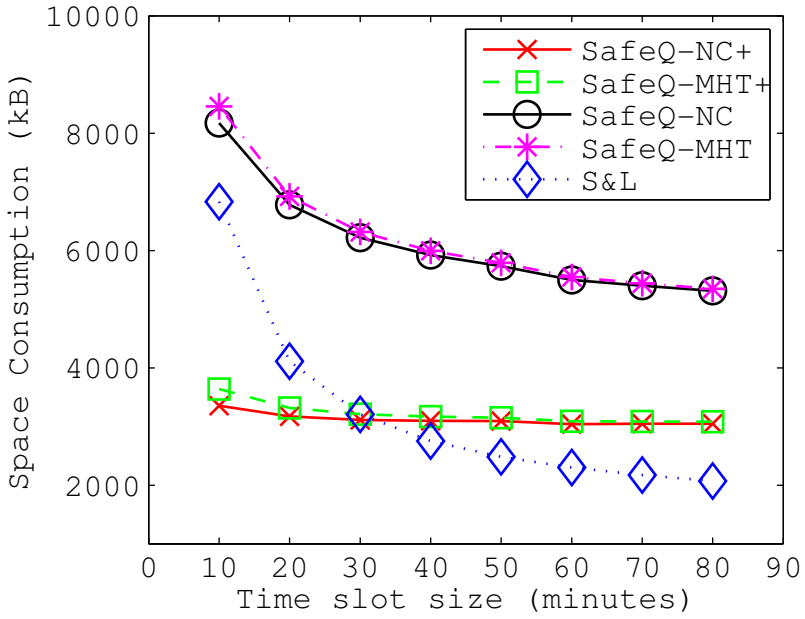


(a) 3-dimensional data

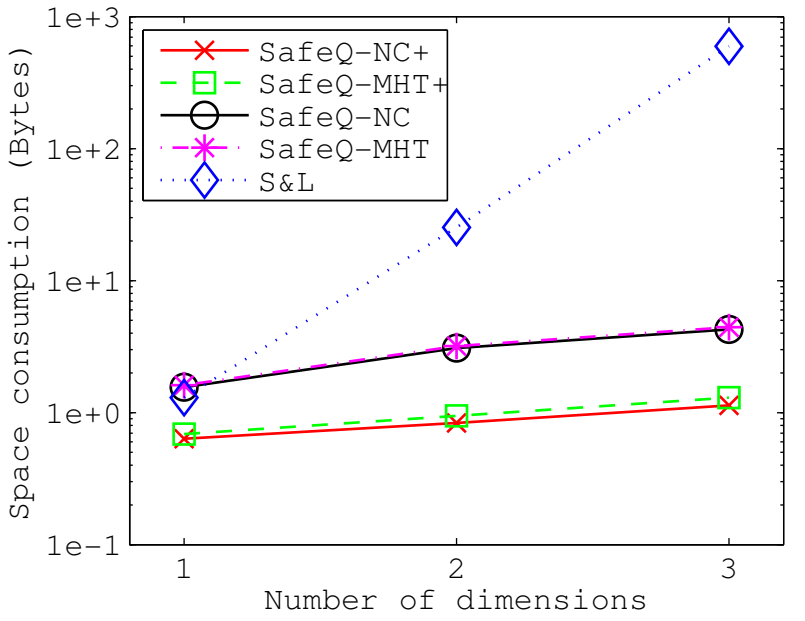


(b) 2-dimensional data

Figure 2.15. Average space consumption for a storage node (A)



(a) 1-dimensional data



(b) Each data item

Figure 2.16. Average space consumption for a storage node (B)

CHAPTER 3

Privacy and Integrity Preserving Range Queries for Cloud Computing

3.1 Introduction

3.1.1 Motivation

Cloud computing has become a new computing paradigm of internet services, where cloud providers host numerous hardware, software, and network resources, to store organizations' data and perform computation over the data on demand of customers' queries. Cloud computing has three major advantages. First, organizations can instantly open business and provide products or services to their customers without building and maintaining their computer infrastructure, which significantly reduces costs. Second, the data stored in a cloud are more reliable and can be accessed whenever a customer has internet connection. Third, cloud providers have powerful computation capacity to process queries, which provides better experience to customers. Many clouds have been successfully built, *e.g.*, Amazon EC2 and S3 [1], Google App Engine [3], and Microsoft Azure [5].

The database-as-a-service (DAS) model, first introduced by Hacigumus *et al.* [40], is one of the most important works in cloud computing. In the DAS model, a cloud provider hosts the data of an organization and replies query results to the customers on behalf of the organization. However, the inclusion of the DAS model brings significant security and privacy challenges. As cloud providers cannot be fully trusted and the data of an organization are typically confidential, the organization needs to encrypt the data before storing it in a cloud to prevent the cloud provider from revealing the data. However, it is difficult to process queries over encrypted data. Furthermore, since cloud providers serve as an important role for answering queries from customers, they may return forged data for the query or may not return all the data items that satisfy the query.

Therefore, we want to design a protocol for the DAS model that supports multi-dimensional range queries while preserving the privacy of both data and queries and the integrity of query results. Range queries are one of the most important queries for various database systems and have wide applications. For data privacy, cloud providers cannot reveal the organization data and customer queries. Note that the customer queries also need to be kept confidential from cloud providers because such queries may leak critical information about query results. For query result integrity, customers need to detect whether a query result includes forged data or does not include all the data items that satisfy the query.

3.1.2 Technical Challenges

There are three challenges in solving secure multi-dimensional range queries problem in the DAS model. First, a cloud provider needs to correctly process range queries over encrypted data without knowing the values of both data and queries. Second, customers need to verify whether a query result contains all the data items that satisfy the query and does not contain any forged data. Third, supporting multi-dimensional range queries is difficult.

3.1.3 Limitations of Previous Work

Privacy and integrity preserving range queries have received much attention in database and security community (*e.g.*, [40, 41, 10, 31, 72, 17, 27, 62, 22]).

Four main techniques have been proposed in the privacy-preserving schemes: bucket partitioning (*e.g.*, [40, 41]), order-preserving hash functions (*e.g.*, [31]), order-preserving encryptions (*e.g.*, [10]), and public-key encryption (*e.g.*, [17, 72]). However, bucket partitioning leads to false positives in query results, *i.e.*, a query result includes data items that do not satisfy the query. Existing order-preserving hash functions and order-preserving encryptions require large amount of shared secret information between an organization and its customers. The public-key cryptography is too expensive to be applied in realistic applications.

Three main techniques have been proposed in the integrity-preserving schemes: Merkle hash trees (*e.g.*, [27, 63]), signature aggregation and chaining (*e.g.*, [62, 59]), and spatial data structures (*e.g.*, [22]). However, Merkle hash trees cannot support multi-dimensional range queries. Signature aggregation and chaining requires a cloud provider to reply to the customer the boundary data items of the query that do not satisfy the query. Spatial data structures are computationally expensive because constructing such structures is complicated. Furthermore, it is not clear how to search query results over such structures in a privacy preserving manner.

3.1.4 Our Approach

In this work, we propose novel privacy and integrity preserving schemes for the DAS model. To preserve privacy, we propose an order-preserving hash-based function to encode both data and queries such that a cloud provider can correctly process encoded queries over encoded data without knowing their values. To preserve integrity, we present the first probabilistic integrity-preserving scheme for multi-dimensional range queries. In this scheme, we propose

a new data structure, called *local bit matrices*, to encode neighborhood information for each data item from an organization, such that a customer can verify the integrity of a query result with a high probability.

Comparing with the state-of-the-art, our schemes achieve both security and efficiency. In terms of security, our schemes not only enable a cloud provider to correctly process queries over encrypted data, but also leak only the minimum privacy information as we will discuss in Section 3.3.4. In terms of efficiency, our schemes are much more efficient due to the use of the hash function and symmetric encryption.

3.1.5 Key Contributions

We make three major contributions. First, we propose an efficient privacy-preserving scheme that can process multi-dimensional range queries without false positives. Second, we propose the first probabilistic scheme for verifying the integrity of range query results. This scheme employs a new data structure, *local bit matrices*, which enables customers to verify query result integrity with high probability. Third, we conduct extensive experiments on real and synthetic datasets to evaluate the effectiveness and efficiency of our scheme.

3.1.6 Summary of Experimental Results

We performed extensive experiments on synthetic datasets and the Adult dataset [32]. Our experimental results show that our schemes are efficient for preserving privacy and integrity of multi-dimensional range queries in cloud computing. For a synthetic dataset with one million 1-dimensional data items, the one-time offline data processing time is about 50 minutes, the space cost is 33MB, and the query processing time is 2 milliseconds. For the Adult dataset with 45222 3-dimensional data items, the data processing time is 104 seconds, the space cost is 1.5MB, and the query processing time is 3.5 milliseconds.

3.2 Models and Problem Statement

3.2.1 System Model

We consider the database-as-a-service (DAS) model as illustrated in Figure 3.1. The DAS model consists of three parties: organizations, a cloud provider, and customers. Organizations outsource their private data to a cloud provider. A cloud provider hosts outsourced data from organizations and processes the customers' queries on behalf of the organizations. Customers are the clients (of organizations) that query a cloud provider and retrieve query results from the outsourced data in the cloud provider.

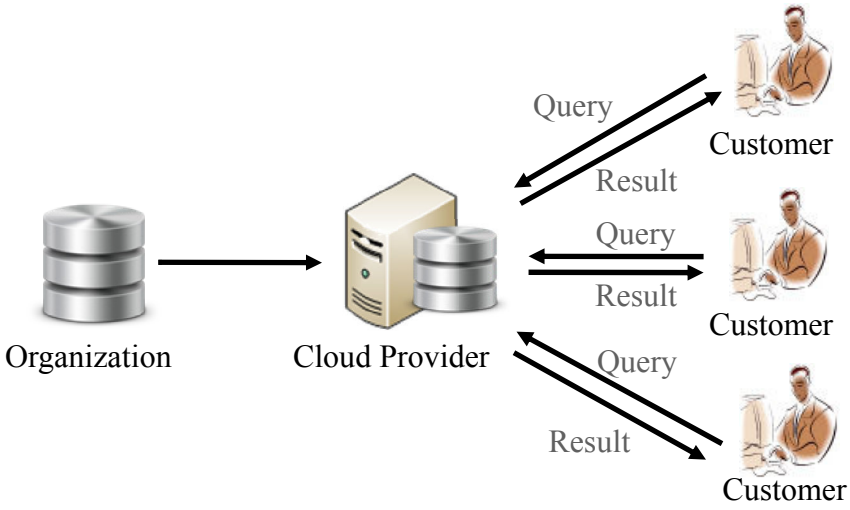


Figure 3.1. The DAS model

3.2.2 Threat Model

In the DAS model, we assume that organizations and their customers are trusted but the cloud provider is not. In a hostile environment, both customers and cloud providers may not be trusted. If a customer is malicious, it may retrieve all the organization's data and distribute to other unauthorized users. Such attack is very difficult to be prevented and is out of the scope of this work. In this work, we mainly focus on the scenario where a cloud

provider is not trusted and it may try to reveal organizations' data and falsify the query results. In reality, cloud providers and organizations typically belong to different parties, *i.e.*, different companies. The organizations cannot share their private data with untrusted cloud providers. A malicious cloud provider may try to reveal the private data of organizations, and return falsified query results that include forged data or exclude legitimate data. In such case, a cloud provider can disrupt the business and cause great lost of organizations.

We also assume that there are secure channels between the organization and the cloud provider, and between the cloud provider and each customer, which could be achieved using protocols such as SSL.

3.2.3 Problem Statement

The fundamental problem for the DAS model is: *how can we design the storage scheme and the query protocol in a privacy and integrity preserving manner*? A satisfactory solution to this problem should meet the following three requirements. (1) *Data and query privacy*: Data privacy means that a cloud provider cannot reveal any data item from organizations. Query privacy means that a cloud provider cannot reveal any query from customers. (2) *Data integrity*: If a cloud provider returns forged data or does not return all the data items that satisfy the query, such misbehavior should be detected by the customer. (3) *Range Query Processing*: The encoded data from organizations and encoded queries from customers should allow a cloud provider to correctly process range queries.

3.3 Privacy Preserving for 1-dimensional Data

In this section, we present our privacy-preserving scheme for 1-dimensional data. To preserve privacy, it is natural to have an organization to encrypt its data items. Let $d_1, \cdots, d_n$ denote n data items from the organization, the encryption results can be denoted as $(d_1)_k, \cdots, (d_n)_k$,

where k is the shared secret key between the organization and its customers. However, the key challenge is how can a cloud provider process queries over encrypted data without knowing the values of data items.

The basic idea of our scheme is to design an order-preserving hash-based function to encode the data items from the organization and the queries from its customers such that the cloud provider can use the encoded queries and encoded data items to find out the query results without knowing the actual values. More formally, let $f_k()$ denote the order-preserving hash-based function, where k is the shared secret key between the organization and its customers. To compute $f_k()$, the organization and its customers also need to share the secret information, the domain of the data items $[x_1, x_N]$. This function $f_k()$ satisfies the following property: the condition $f_k(x_{i_1}) < f_k(x_{i_2})$ holds if $x_1 \leq x_{i_1} < x_{i_2} \leq x_N$. To submit n data items $d_1, \cdots, d_n$ to a cloud provider, the organization first encrypts each data item with the secret key k, *i.e.*, $(d_1)_k, \cdots, (d_n)_k$. Second, the organization applies the function $f_k()$ to each data item, *i.e.*, $f_k(d_1), \cdots, f_k(d_n)$. Finally, the organization sends the encrypted data items $(d_1)_k, \cdots, (d_n)_k$ as well as the encoded data items $f_k(d_1), \cdots, f_k(d_n)$ to the cloud provider. To perform a range query $[a, b]$, the customer applies the order-preserving hash-based function $f_k()$ to the lower and upper bounds of the query, *i.e.*, $f_k(a)$ and $f_k(b)$, and then sends $[f_k(a), f_k(b)]$ as the query to the cloud provider. Finally, the cloud provider can find out whether the data item d_j $(1 \leq j \leq n)$ satisfies the query $[a, b]$ by checking whether the condition $f_k(a) \leq f_k(d_j) \leq f_k(b)$ holds. Figure 3.2 shows the idea of our privacy-preserving scheme.

In this section, we first present our order-preserving hash-based function and then discuss its properties. Second, we propose the privacy-preserving scheme by employing this function. Third, we propose an optimization technique to reduce the size of the results after applying the hash-based function. Fourth, we analyze the minimum information leakage for any precise privacy-preserving scheme of range queries and demonstrate that our scheme leaks

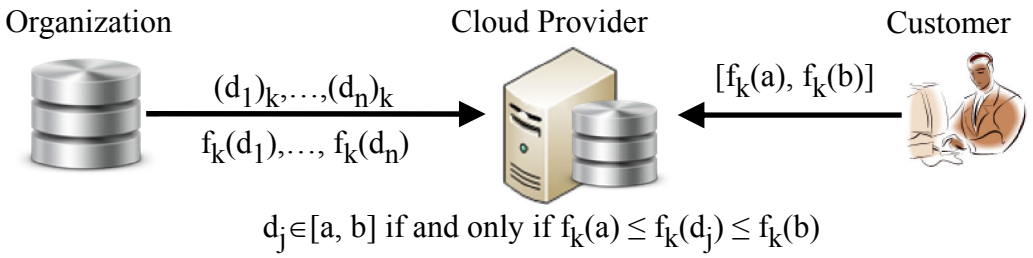


Figure 3.2. Basic idea of privacy-preserving scheme

only the minimum information.

3.3.1 The Order-Preserving Hash-based Function

Without loss of generalization, we assume that all possible data items are integers within domain $[x_1, x_N]$. The order-preserving hash-based function $f_k()$ is in the form

$$f_k(x_i) = \sum_{q=1}^{j} h_k(x_q) \tag{3.1}$$

where $x_i \in [x_1, x_N]$ and $h_k()$ is a keyed hash function, such as keyed HMAC-MD5 and keyed HMAC-SHA1. The intuition of designing such order-preserving hash-based function is two-fold. First, we leverage a normal hash function $h_k()$ as the basic building block such that the one-way property of $h_k()$ prevents the cloud provider from revealing the data items. Second, we consider the result of $h_k()$ as a positive integer and then calculate $f_k(x_i)$ by summing the hash results of all values that are less than or equal to x_i in the domain $[x_1, x_N]$ such that if $x_1 \leq x_{i_1} < x_{i_2} \leq x_N$, $f_k(x_{i_1})$ is less than $f_k(x_{i_2})$. In other words, $f_k()$ is an order-preserving function for the values in $[x_1, x_N]$.

More formally, the order-preserving hash-based function $f_k()$ satisfies two properties.

Order Preserving: *Assume that any* $h_k(x_q)$ $(x_q \in [x_1, x_N])$ *is a positive integer. The condition* $f_k(x_{i_1}) < f_k(x_{i_2})$ *holds if and only if* $x_{i_1} < x_{i_2}$.

Proof. We first prove that if the condition $f_k(x_{i_1}) < f_k(x_{i_2})$ holds, then $x_{i_1} < x_{i_2}$. We prove it by contradiction. If $x_{i_1} \geq x_{i_2}$, we have

$$f_k(x_{i_1}) = f_k(x_{i_2}) + \sum_{q=i_2+1}^{i_1} h_k(x_q) \geq f_k(x_{i_2}) \tag{3.2}$$

Second, we prove that if the condition $x_{i_1} < x_{i_2}$ holds, then $f_k(x_{i_1}) < f_k(x_{i_2})$. Similar as the proof of the property collision resistance, we have

$$f_k(x_{i_2}) = f_k(x_{i_1}) + \sum_{q=i_1+1}^{i_2} h_k(x_q) > f_k(x_{i_1})$$

□

Collision Resistance: *Assume that any $h_k(x_q)$ ($x_q \in [x_1, x_N]$) is a positive integer. It is impossible to find x_{i_1} and x_{i_2} where $x_{i_1} \neq x_{i_2}$ such that $f_k(x_{i_1}) = f_k(x_{i_2})$.*

Proof. Without loss of generalization, we assume $i_1 < i_2$. Because any $h_k(x_q)$ ($x_q \in [x_1, x_N]$) is a positive integer, we have

$$f_k(x_{i_2}) = f_k(x_{i_1}) + \sum_{q=i_1+1}^{i_2} h_k(x_q) > f_k(x_{i_1})$$

□

In fact, the hash-based function $f_k()$ can preserve any given *arbitrary order* of values in the domain $[x_1, x_N]$ no matter whether the condition $x_1 < \cdots < x_N$ holds. For example, if the order of 3 data items $3, 5, 7$ is defined as $5, 3, 7$, then $f_k(5) < f_k(3) < f_k(7)$. This property allows an organization to revise any data item x_i ($x_i \in [x_1, x_N]$) arbitrarily while still preserving the order. We will discuss how to leverage this property to prevent the statistical analysis of multi-dimensional data in Section 3.6.1.

These two properties and the one-way property of $h_k()$ allow the cloud provider to process the encoded range queries over the encoded data without revealing the values of the data and queries.

3.3.2 The Privacy-Preserving Scheme

The privacy-preserving scheme includes three phases, *data submission*, *query submission*, *query processing*.

The data submission phase concerns how an organization sends its data to a cloud provider. Let $d_1, \cdots, d_n$ denote the data items of an attribute in the private data of the organization. Recall that $[x_1, x_N]$ is the domain of the attribute and is the shared secret between the organization and its customers. For simplicity, we assume $d_1 < d_2 < \cdots < d_n$. If some data items have the same value, the organization can simply represent them as one data item annotated with the number of items that share this value.

To preserve data privacy, for each d_j $(1 \leq j \leq n)$, the organization first encrypts it with its secret key k, *i.e.*, $(d_j)_k$, and then applies the order-preserving hash-based function, *i.e.*, $f_k(d_j)$. Finally, the organization sends the encrypted data $(d_1)_k, \cdots, (d_n)_k$ as well as the hash results $f_k(d_1), \cdots, f_k(d_n)$ to the cloud provider.

The query submission phase concerns how a customer sends a range query to the cloud provider. When a customer wants to perform a range query $[a, b]$ on the cloud provider, it first applies the order-preserving hash-based function to the lower and upper bounds of the query, *i.e.*, $f_k(a)$ and $f_k(b)$. Note that a and b are also two values in $[x_1, x_N]$. Finally, the customer sends $[f_k(a),\ f_k(b)]$ as a query to the cloud provider.

Upon receiving the query $[f_k(a),\ f_k(b)]$, the cloud provider processes this query on the n data items $(d_1)_k, \cdots, (d_n)_k$ by checking which $f_k(d_j)$ $(1 \leq j \leq n)$ satisfies the condition $f_k(a) \leq f_k(d_j) \leq f_k(b)$. Based on the order preserving property of the function $f_k()$, $d_j \in [a, b]$ if and only if $f_k(a) \leq f_k(d_j) \leq f_k(b)$. Thus, the cloud provider only needs to return all encrypted data items whose hash values $f_k()$ are in the range $[f_k(a),\ f_k(b)]$.

3.3.3 Optimization of the Order-Preserving Hash-based Function

We propose an optimization technique to reduce the communication cost for sending the encoded data $f_k(d_1), \cdots, f_k(d_n)$ to the cloud provider. This cost can be significant for two reasons. First, the result of the hash function $h_k()$ is long. For example, the result of the keyed HMAC-MD5 is 128 bits, and the result of the keyed HMAC-SHA1 is 160 bits. Second, the result of our order-preserving hash-based function $f_k()$ is even longer due to the sum of multiple hash values of $h_k()$. Let w denote the bit length of $h_k()$. For any x_i ($x_i \in [x_1, x_N]$), the condition $h_k(x_i) \leq 2^w - 1$ holds. Thus, we have

$$f_k(x_i) \leq f_k(x_N) = \sum_{q=1}^{N} h_k(x_q) \leq \sum_{q=1}^{N} (2^w - 1) < 2^w N \tag{3.3}$$

Therefore, the bit length of $f_k(x_i)$ ($1 \leq i \leq N$) is less than $\log_2 2^w N = w + \log_2 N$. Assume that we use the keyed HMAC-MD5 as the hash function $h_k()$ and the number of possible values is one million, *i.e.*, $N = 10^6$. Any $f_k(x_i)$ can be expressed by a $128 + \log_2 10^6 = 148$ bit number.

To reduce the bit length of $f_k(x_i)$ ($x_i \in [x_1, x_N]$), the idea is to divide every $f_k(x_i)$ by a value $2^{w'}$ if $2^{w'}$ is less than or equal to any $h_k(x_q)$ ($x_q \in [x_1, x_N]$). Then, our order-preserving hash-based function becomes

$$f_k^*(x_i) = \frac{\sum_{q=1}^{i} h_k(x_q)}{2^{w'}} \tag{3.4}$$

where $2^{w'} \leq h_k(x_q)$ for any $x_q \in [x_1, x_N]$. Such division can be easily done by deleting the right w' bits of $f_k(x_i)$.

Similar as the analysis of the bit length of $f_k(x_i)$, the bit length of any $f_k^*(x_i)$ ($x_i \in [x_1, x_N]$) can be reduced to $w - w' + log_2 N$ by the following calculation.

$$f_k^*(x_i) \leq f_k^*(x_N) \leq \frac{\sum_{q=1}^{N} (2^w - 1)}{2^{w'}} < (2^{w-w'})N \tag{3.5}$$

The order-preserving function $f_k^*()$ also satisfies the two properties, order preserving and collision resistance, the proof of which is in Appendix B.

3.3.4 Analysis of Information Leakage

Given n data items $d_1, \cdots, d_n$ and a range query $[a, b]$, any precise privacy-preserving scheme should enable the cloud provider to find out all the data items that satisfy the query $[a, b]$ without revealing the values of the data items from the organization and the query from its customer. According to this requirement, we have the following theorem.

Theorem 2. *Given any precise privacy-preserving scheme, if all possible results of range queries have been found during the query processing phase, the cloud provider can reveal the order of the encrypted data items.*

Proof. Without loss of generalization, we assume $d_1 < d_2 < \cdots < d_n$. First, we show that how to reveal the order of three consecutive data items d_{j-1}, d_j, d_{j+1}. Among all possible results of range queries, there should be two query results, $QR_1 = \{(d_{j-1})_k, (d_j)_k\}$ and $QR_2 = \{(d_j)_k, (d_{j+1})_k\}$. Assume that $[a_1, b_1]$ and $[a_2, b_2]$ are the two range queries whose results are QR_1 and QR_2, respectively. Obviously, $[a_1, b_1]$, $[a_2, b_2]$, and d_{j-1}, d_j, d_{j+1} satisfy two conditions.

1. $d_{j-2} < a_1 \leq d_{j-1} < d_j \leq b_1 < d_{j+1}$.
2. $d_{j-1} < a_2 \leq d_j < d_{j+1} \leq b_2 < d_{j+2}$.

Based on QR_1, the cloud provider knows that $(d_{j-1})_k$ and $(d_j)_k$ are two consecutive data items. Similarly, based on QR_2, the cloud provider knows that $(d_j)_k$ and $(d_{j+1})_k$ are two consecutive data items. Based on the common encrypted data item $(d_j)_k$ in QR_1 and QR_2, the cloud provider knows that d_j is between d_{j-1} and d_{j+1}. Thus, the cloud provider knows

the order of these three encrypted data items is either $d_{j-1} < d_j < d_{j+1}$ or $d_{j-1} > d_j > d_{j+1}$. Repeat sorting other three consecutive items. Finally, the cloud provider knows the order of all the encrypted data items is either $d_1 < \cdots < d_n$ or $d_1 > \cdots > d_n$. □

Theorem 2 describes the minimum information leakage for any precise privacy-preserving scheme of range queries. That is, the cloud provider will reveal the order of the data items received from the organization.

We argue that our privacy-preserving scheme achieves the minimum information leakage for two reasons. First, the cloud provider cannot reveal the values of the data and queries from the hash results due to the one-way property of $h_k()$. Second, the cloud provider cannot reveal these values by launching statistical analysis because for any two data items d_{j_1} and d_{j_2} $(1 \leq j_1 < j_2 \leq n)$, $(d_{j_1})_k \neq (d_{j_2})_k$ and $f_k(d_{j_1}) \neq f_k(d_{j_2})$. Recall that if some data items have the same value, the organization represents them as one data item with the number of items that share this value.

3.4 Integrity Preserving for 1-dimensional Data

In this section, we present the first probabilistic integrity-preserving scheme for 1-dimensional data. This scheme allows a customer to verify the integrity of a query result with a high probability. The meaning of integrity preserving is two-fold. First, a customer can verify whether the cloud provider forges some data items in the query result. Second, a customer can verify whether the cloud provider deletes data items that satisfies the query.

The basic idea of the integrity-preserving scheme is to encrypt neighborhood information for each data item such that the neighborhood information of the data items in a query result can be used to verify the integrity of the query result. More formally, let $(M(d_j))_k$ denote the encrypted neighborhood information for each data item d_j $(1 \leq j \leq n)$. To submit n data items $d_1, \cdots, d_n$ to a cloud provider, the organization not only sends the

encrypted data items $(d_1)_k, \cdots, (d_n)_k$ and the encoded data items $f_k(d_1), \cdots, f_k(d_n)$, but also sends the encrypted neighborhood information $(M(d_1))_k, \cdots, (M(d_n))_k$. Upon receiving a query $[f_k(a), f_k(b)]$ from a customer, the cloud provider first finds out the query result based on the privacy-preserving scheme. Suppose that the data items $d_{j_1}, \cdots, d_{j_2}$ $(1 \leq j_1 \leq j_2 \leq n)$ satisfy the query. The cloud provider not only replies to the customer the query result $(d_{j_1})_k, \cdots, (d_{j_2})_k$, but also replies the encrypted neighborhood information $(M(d_{j_1}))_k, \cdots, (M(d_{j_2}))_k$. For ease of presentation, let QR denote the *query result*, which includes all the encrypted data items that satisfy the query, *i.e.*, $QR = \{(d_{j_1})_k, \cdots, (d_{j_2})_k\}$, and VO denote the *verification object*, which includes the information for the customer to verify the integrity of QR, *i.e.*, $VO = \{(M(d_{j_1}))_k, \cdots, (M(d_{j_2}))_k\}$. To verify the integrity, the customer first decrypts the query result and verification object, *i.e.*, computes $d_{j_1}, \cdots, d_{j_2}$ and $M(d_{j_1}), \cdots, M(d_{j_2})$. Second, the organization checks whether $d_{j_1}, \cdots, d_{j_2}$ satisfy the query and the overlapping parts of the neighborhood information from every two adjacent data items exactly match. If so, the customer concludes that the query result includes all the data items that satisfy the query. Otherwise, the customer concludes that some data items in the query result are forged or deleted by the cloud provider. Figure 3.3 shows the basic idea of our integrity-preserving scheme.

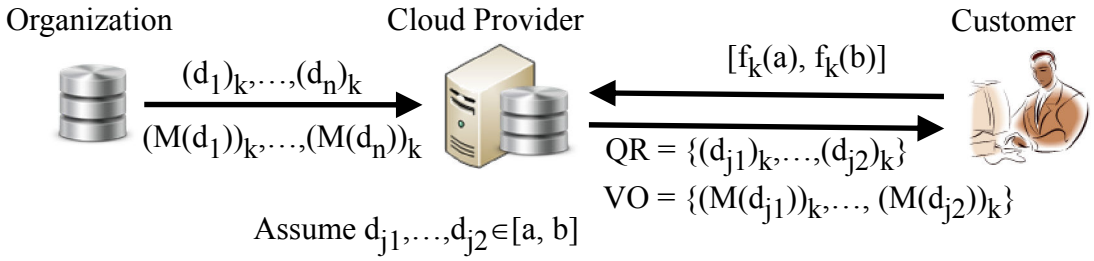


Figure 3.3. Basic idea of integrity-preserving scheme

Our integrity-preserving scheme can guarantee to detect the misbehavior of forging data items because the cloud provider cannot insert fake data items into a query result without knowing the secret key k. This scheme also allows a customer to detect the misbehavior of

deleting data items in a query result with a high probability.

Next we present new data structures, called *bit matrices* and *local bit matrices*, and then discuss their usage in integrity verification.

3.4.1 Bit Matrices and Local Bit Matrices

To define bit matrices and local bit matrices, we need to first partition the data domain into multiple non-overlapping buckets. For example in Figure 3.4, we partition domain $[1, 15]$ to five buckets, $B_1 = [1, 3]$, $B_2 = [4, 6]$, $B_3 = [7, 10]$, $B_4 = [11, 12]$, and $B_5 = [13, 15]$. Second, we distribute the data items into the corresponding buckets. Third, we assign a bit value 1 to the buckets which includes data items, and assign a bit value 0 to the buckets which does not include data items. Let $B(d_j)$ denote the bucket that includes d_j. A bucket is called the *left nonempty bucket* of data item d_j if the bucket is the left nearest bucket of $B(d_j)$ that includes data items. Similarly, a bucket is called the *right nonempty bucket* of data item d_j if the bucket is the right nearest bucket of $B(d_j)$ that includes data items. For example, for data item 7 in Figure 3.4, B_2 and B_5 are the left and right nonempty buckets of data item 7, respectively.

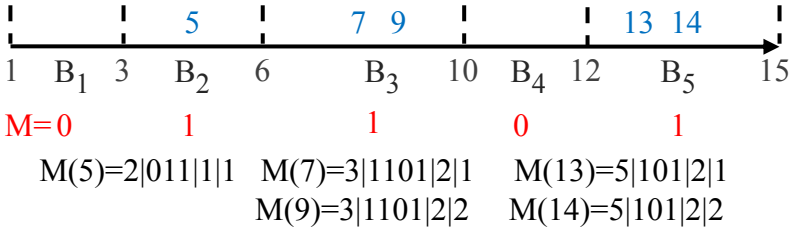


Figure 3.4. Example bit matrix and local bit matrices

Based on the above concepts, we define bit matrices and local bit matrices as follows. The *bit matrix* of all data items, M, is formed by the bit values of all buckets. In Figure 3.4, the bit matrix of the five data items is 01101, *i.e.*, $M = 01101$. The *local bit matrix* of a data item d_j, $M(d_j)$, consists of four parts: (1) the bucket id of $B(d_j)$; (2) a subset of the bit

matrix, which is formed by the bit values from its left nonempty bucket to its right nonempty bucket; (3) the number of data items in bucket $B(d_j)$; (4) a distinct integer to distinguish the local bit matrix of d_j from other data items in bucket $B(d_j)$. In Figure 3.4, the local bit matrix of data item 7 is 3|1101|2|1, *i.e.*, $M(7) = 3|1101|2|1$, where 3 is the bucket id, 1101 is the subset of the bit matrix, 2 is the number of data items in bucket B_3, and 1 is the integer to distinguish $M(7)$ from $M(9)$. Intuitively, the bit matrix denotes the abstract information of all the data items, and the local bit matrix of a data item d_j denotes the abstract neighborhood information of d_j.

Note that the usage of bucket partition in this work is different from that in previous work (*e.g.*, [40, 41, 10]). They leverage bucket partition to achieve privacy-preserving query processing. While we use the bit values of buckets for verifying the integrity of query results.

3.4.2 The Integrity-Preserving Scheme

Our integrity-preserving scheme includes four phases, *data submission*, *query submission*, *query processing*, and *query result verification*.

Let $d_1, \cdots, d_n$ denote the data items of an attribute from the organization. The organization first partitions the data domain to m non-overlapping buckets $B_1, \cdots, B_m$, and then distributes $d_1, \cdots, d_n$ to these buckets. The bucket partition is a shared secret between the organization and its customers. Second, the organization computes the local bit matrix for each data item and then encrypts them with its secret key k, *i.e.*, computes $(M(d_1))_k, \cdots, (M(d_n))_k$. Third, the organization sends to the cloud provider the encrypted local bit matrices $(M(d_1))_k, \cdots, (M(d_n))_k$ as well as encrypted data items $(d_1)_k, \cdots, (d_n)_k$ and the encoded data items $f_k(d_1), \cdots, f_k(d_n)$.

To perform a range query $[a, b]$, a customer sends $[f_k(a),\ f_k(b)]$ to the cloud provider.

Upon receiving $[f_k(a),\ f_k(b)]$, the cloud provider computes QR as in Section 3.3.2. Here

we consider how to compute VO. If $QR = \{(d_{j_1})_k, \cdots, (d_{j_2})_k\}$ $(1 \leq j_1 \leq j_2 \leq n)$, $VO = \{(M(d_{j_1}))_k, \cdots, (M(d_{j_2}))_k\}$; if $QR = \emptyset$, which means that there is a data item d_{j_1} $(1 \leq j_1 \leq n)$ such that $d_{j_1} < a \leq b < d_{j_1+1}$, then $VO = \{(M(d_{j_1}))_k, (M(d_{j_1+1}))_k\}$. Finally, the cloud provider replies QR and VO.

Upon receiving the query result QR and the verification object VO, the customer decrypts them, and then verifies the integrity of QR as follows. First, the customer verifies whether each item in QR satisfies the query $[a, b]$. Second, the customer verifies whether the cloud provider deletes any data item that satisfies the query. Let $\{(d_{j_1})_k, \cdots, (d_{j_2})_k\}$ be the correct query result and $B_{g_1}, \cdots, B_{g_t}$ be the buckets which include at least one data item in the query result. Let QR be the query result from the cloud provider. Suppose the cloud provider deletes a data item $(d_j)_k$ that satisfies the query, *i.e.*, $(d_j)_k \in \{(d_{j_1})_k, \cdots, (d_{j_2})_k\}$, and $d_j \in B_{g_s}$ $(1 \leq s \leq t)$. We consider the following four cases.

***Case* 1**: When $QR \neq \emptyset$, if $B_{g_s} \subseteq [a, b]$, the deletion can be detected for two reasons. First, if B_{g_s} only includes one data item d_j, deleting $(d_j)_k$ can be detected because the local bit matrices of data items in $B_{g_{s-1}}$ or $B_{g_{s+1}}$ show that B_{g_s} should include at least one data item. Second, if B_{g_s} includes multiple data items, deleting $(d_j)_k$ can be detected because the local bit matrices of other data items in B_{g_s} have the number of data items in B_{g_s}. In Figure 3.4, given a range query [4,11], the correct query result is $\{(5)_k, (7)_k, (9)_k\}$, and the verification object is $\{(M(5))_k, (M(7))_k, (M(9))_k\}$. Deleting $(7)_k$ in B_3 can be detected because based on $M(9)$, the customer knows that B_3 includes two data items.

***Case* 2**: When $QR \neq \emptyset$, if $B_{g_s} \not\subseteq [a, b]$, the deletion cannot be detected because the customer does not know whether $B_{g_s} \cap [a, b]$ includes data items. Considering the same example in Case 1, deleting $(5)_k$ cannot be detected because the customer does not know whether $B_2 \cap [4, 11]$ includes data items.

***Case* 3**: When $QR = \emptyset$, if $B(d_{j_1}) \cap [a, b] = \emptyset$ and $B(d_{j_1+1}) \cap [a, b] = \emptyset$, the deletion can be detected because $M(d_{j_1})$ or $M(d_{j_1+1})$ shows that bucket B_{g_s} between $B(d_{j_1})$ and $B(d_{j_1+1})$

includes data items, and hence, condition $d_{j_1} < a \leq b < d_{j_1+1}$ does not hold. In Figure 3.4, given a range query [3,5], the correct query result is $\{(5)_k\}$. If the cloud provider replies $QR = \emptyset$ and $VO = \{(M(7))_k\}$, deleting $(5)_k$ can be detected because the customer knows that B_2 includes data items based on $M(7)$, and these data items are closer to the query [3,5] than 7.

***Case* 4**: When $QR = \emptyset$, if $B(d_{j_1}) \cap [a, b] \neq \emptyset$ or $B(d_{j_1+1}) \cap [a, b] \neq \emptyset$, the deletion cannot be detected because the customer does not know whether $B(d_{j_1}) \cap [a, b]$ or $B(d_{j_1+1}) \cap [a, b]$ includes data items. In Figure 3.4, given a range query [9,12], the correct query result is $\{(9)_k\}$. If the cloud provider replies $QR = \emptyset$ and $VO = \{(M(7))_k, (M(13))_k\}$, deleting $(9)_k$ cannot be detected because the customer does not know whether $B(3) \cap [9, 12]$ includes data.

The cloud provider can break integrity verification if and only if it can distinguish Cases 2 and 4 from other two cases. Distinguishing these two cases is equivalent to knowing which data items belong to the same bucket. However, such information cannot be revealed by analyzing the encrypted data items $(d_1)_k, \cdots, (d_n)_k$, the encoded data items $f_k(d_1), \cdots, f_k(d_n)$, and the encrypted local bit matrices $(M(d_1))_k, \cdots, (M(d_n))_k$. Because for any two data items d_{j_1} and d_{j_2} $(1 \leq j_1 < j_2 \leq n)$, $(d_{j_1})_k \neq (d_{j_2})_k$, $f_k(d_{j_1}) \neq f_k(d_{j_2})$, and $(M(d_{j_1}))_k \neq (M(d_{j_2}))_k$. Thus, the cloud provider can only randomly delete the data items in the query result and hope that this deletion operation will not be detected.

3.5 Finding Optimal Parameters

Our integrity-preserving scheme needs to partition the data domain into multiple non-overlapping buckets. However, how to partition the domain is still a problem. In this section, we formalize the problem as an optimization problem and present an algorithm to solve it. To simplify the problem, we assume that queries from customers follow uniform distribution, *i.e.*, all queries are equi-probable. Considering the N possible values in $[x_1, x_N]$,

there are $\frac{N(N+1)}{2}$ possible range queries. Thus, the possibility of any query from customers is equal to $\frac{2}{N(N+1)}$.

3.5.1 Detection Probability

We first consider the *detection probability* of randomly deleting data items in a query result by a cloud provider. This metric is very important to evaluate the effectiveness of our integrity-preserving scheme. Let $B_1, \cdots, B_m$ denotes the multiple non-overlapping buckets, and $[l_i, h_i]$ denotes a bucket B_i $(1 \leq i \leq m)$. A bucket B_i is called a *single-value bucket* if $l_i = h_i$. Let $e(x)$ denote the frequency of the data item with value x. The probability that a deletion operation of the cloud provider can be detected is

$$Pr = \frac{\sum_{i=1}^{m} \sum_{x=l_i}^{h_i} (l_i - x_1 + 1)(x_N - h_i + 1)e(x)}{\sum_{j=1}^{n} (d_j - x_1 + 1)(x_N - d_j + 1)} \tag{3.6}$$

The calculation of this probability is in Appendix C. Next, we discuss the two theorems regarding to Pr, the proofs of these two theorems are in Appendix D.

Theorem 3. *Given any n data items $d_1, \cdots, d_n$, the maximum detection probability of a deletion operation is*

$$Pr_{max} = 100\% \tag{3.7}$$

if and only if each data item d_j $(1 \leq j \leq n)$ forms a single-value bucket, i.e., $[d_j, d_j]$.

Theorem 4. *Given n data items $d_1, \cdots, d_n$, the minimum detection probability of a deletion operation is*

$$Pr_{min} = \frac{n}{\sum_{j=1}^{n} (d_j - x_1 + 1)(x_N - d_j + 1)} \tag{3.8}$$

if and only if there is only one bucket $[x_1, x_N]$.

The intuition behind Theorems 3 and 4 is that the more secret information of data items the organization shares with its customers, the higher detection probability customers

achieve. For Theorem 3, if each data item forms a single-value bucket, the customers know all the data items before querying the cloud provider. Of course they can detect any deletion operation and it is meaningless for organizations to outsource their data. For Theorem 4, recall our privacy preserving scheme in Section 3.3. Customers need to know $[x_1, x_N]$ for converting a query $[a, b]$ to $[f_k(a), f_k(b)]$. Thus, in our context, x_1 and x_N are the minimum secret information needed to be shared. Knowing only x_1 and x_N allows customers to detect deletion operations with the minimum probability.

If a cloud provider conducts t deletion operations, the probability that at least one of the t deletion operations can be detected is

$$Pr_t = 1 - (1 - Pr)^t \tag{3.9}$$

In Figure 3.4, the probability that a deletion operation can be detected is 60.48%. If the cloud provider conducts 5 deletion operations over the query results, customers can detect at least one deletion with 99% probability.

3.5.2 Optimal Bucket Partition

We define the optimization problem as follows.

Given n data items from the organization and the domain of these items $[x_1, x_N]$, we want to find out the optimal partition with at most m buckets $B_1, \cdots, B_m$ such that the detection probability Pr is maximized. More formally, this problem can be defined as follows.

$$\begin{aligned} \text{Input: } & (1)\ d_1, \cdots, d_n \\ & (2)\ [x_1, x_N] \\ & (3)\ m \\ \text{Output: } & B_1, \cdots, B_m \\ \text{Objective: } & \max Pr \end{aligned}$$

This problem has the optimal substructure property [25]. Therefore, we can express the optimal solution of the original problem as the combination of the optimal solutions of

two sub-problems. Let $H(N, m)$ denote the problem of optimally partitioning the domain $[x_1, x_N]$ using at most m buckets. Let $\delta(i, j)$ denote the probability contributed by a bucket $[x_i, x_j]$. We can compute $\delta(i, j)$ as follows.

$$\delta(i,j) = \frac{(x_i - x_1 + 1)(x_N - x_j + 1)\sum_{x=x_i}^{x_j} e(x)}{\sum_{j=1}^{n}(d_j - x_1 + 1)(x_N - d_j + 1)} \tag{3.10}$$

The optimal problem can be expressed as follows.

$$H(N, m) = \max\left[H(N - i, m - 1) + \delta(N - i + 1, N)\right] \tag{3.11}$$

Algorithm 1: **Optimal Bucket Partition**

Input: (1) n data items $d_1, \cdots, d_n$;

(2) The domain $[x_1, x_N]$;

(2) m.

Output: $B_1, \cdots, B_m$.

```
1 Initialize each element in matrices H and P to 0;
2 for i := 2 to N do
3     H[i][2] = max_{1≤k≤i−1} [δ(1, k) + δ(k + 1, i)];
4     Store the left boundary value of the second bucket in P[i][2];
5 for j := 3 to m do
6     for i := j to N do
7         H[i][j] = max_{j−1≤k≤i−1} [H[k][j − 1] + δ(k + 1, i)];
8         Store the left boundary value of the last bucket in P[i][j];
9 Find the maximum value in H and output the corresponding partition in P;
```

We use dynamic programming to solve the problem. We first solve and store solutions of the smaller sub-problems. Then, we employ their optimal solutions to solve the larger

problems. Finally, we solve the optimal problem of maximizing $H(N, m)$. All intermediate solutions are stored in an $N \times m$ matrix H. The row indices of H are from $1, \cdots, N$ and the column indices are from $1, \cdots, m$. Note that $H(i, j) = H[i][j]$. Both the time and space complexities of the computation of the matrix H are $O(Nm)$. Along with the optimal value $H(i, j)$ $(1 \leq i \leq N, 1 \leq j \leq m)$, we also store the lower bounds of its last bucket for each sub-problem in another $N \times m$ matrix P . Finally, we use the matrix P to reconstruct the optimal bucket partition in $O(m)$ time. This algorithm is shown in Algorithm 1.

3.6 Query Over Multi-dimensional Data

3.6.1 Privacy for Multi-dimensional Data

Organizations' data and customers' queries are typically multi-dimensional. For example, a medical record typically includes patient's name, birthday, age, *etc.* A z-dimensional data item D is a z-tuple $(d^1, \cdots, d^z)$ where each d^r $(1 \leq r \leq z)$ is the value for the r-th dimension (*i.e.*, attribute). A z-dimensional range query consists of z sub-queries $[a^1, b^1], \cdots, [a^z, b^z]$ where each sub-query $[a^r, b^r]$ $(1 \leq r \leq z)$ is a range over the r-th dimension.

We extend our privacy-preserving scheme for one-dimensional data to multi-dimensional data as follows. Let $D_1, \cdots, D_n$ denote n z-dimensional data items, where $D_j = (d_j^1, \cdots, d_j^z)$ $(1 \leq j \leq n)$. First, the organization encrypts these data with its secret key k, *i.e.*, computes $(D_1)_k, \cdots, (D_n)_k$. Second, for each dimension r, it applies our order-preserving hash-based function $f_{k_r}()$, *i.e.*, computes $f_{k_r}(d_1^r), \cdots, f_{k_r}(d_n^r)$, where k_r is the secret key of the order-preserving hash-based function for the r-th dimension. Last, it sends the encrypted data items $(D_1)_k, \cdots, (D_n)_k$, and $f_{k_1}(d_1^1), \cdots, f_{k_1}(d_n^1)$, $\cdots$, $f_{k_z}(d_1^z), \cdots, f_{k_z}(d_n^z)$ to the cloud provider. When a customer wants to perform a query $([a^1, b^1], \cdots, [a^z, b^z])$, it applies the order-preserving hash-based function $f_{k_r}()$ on the lower and upper bounds of each sub-query

$[a^r, b^r]$ and sends $[f_{k_1}(a^1), f_{k_1}(b^1)], \cdots, [f_{k_z}(a^z), f_{k_z}(b^z)]$ to the cloud provider. The cloud provider then compares $f_{k_r}(d_1^r), \cdots, f_{k_r}(d_n^r)$ with $[f_{k_r}(a^r), f_{k_r}(b^r)]$ for each dimension r, to find out the query result QR. Considering 5 two-dimensional data items (1,11), (3,5), (6,8), (7,1) and (9,4), given a range query ([2,7],[3,8]), the query result QR is $\{(3,5)_k, (6,8)_k\}$.

To prevent the attack of statistical analysis, the data sent from the organization to the cloud provider should satisfy the following two conditions. First, for any $1 \leq j_1 \neq j_2 \leq n$, $(D_{j_1})_k \neq (D_{j_2})_k$. To satisfy this condition, if multiple data items have the same value for each dimension, the organization can simply represent them as one data item annotated with the number of these items. Second, along each dimension r, for any $1 \leq j_1 \neq j_2 \leq n$, $f_{k_r}(d_{j_1}^r) \neq f_{k_r}(d_{j_2}^r)$. To satisfy this condition, the organization needs to revise the data items with the same value for the dimension r. Recall the arbitrary order-preserving property of $f_{k_r}()$. It allows the organization to arbitrarily revise data items while still preserving the order of these items in the hash results. In our context, if $d_{j_1}^r = d_{j_2}^r$, the organization can concatenate a distinct number for each of them, *i.e.*, $d_{j_1}^r|0$ and $d_{j_2}^r|1$, and then apply the hash-based function $f_{k_r}()$.

3.6.2 Integrity for Multi-dimensional Data

To preserve the integrity of multi-dimensional data, the organization builds multi-dimensional local bit matrices. We first present the data structures, multi-dimensional bit matrices and local bit matrices, and then discuss the usage in integrity verification for multi-dimensional data. Considering the example in Figure 3.5(a), we partition the data domain into $4 \times 6 = 24$ buckets. Then, we distribute the data items, $D_1, \cdots, D_5$, into the corresponding buckets. We assign a bit value 1 or 0 to each bucket to indicate whether the bucket includes data items or not. Let $B(d_j)$ denote the bucket that includes d_j. A bucket is called the *r-th left nonempty bucket* of data item d_j $(1 \leq r \leq z)$ if the bucket is the left nearest

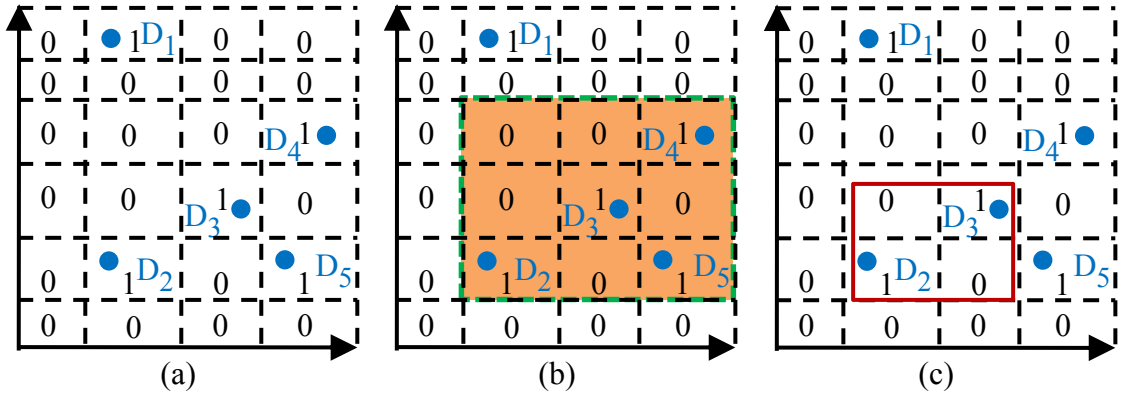


Figure 3.5. The example 2-dimensional bit matrix and local bit matrices

bucket of $B(d_j)$ that includes data items for the r-th dimension. Similarly, a bucket is called the *r-th right nonempty bucket* of data item d_j if the bucket is the right nearest bucket of $B(d_j)$ that includes data items for the r-th dimension. In Figure 3.5(a), $B(D_2)$ is the 1-th left nonempty bucket of data item D_3.

Based on the above concepts, we define bit matrices and local bit matrices as follows. The *bit matrix* of all data items, M, is formed by the bit values of all buckets. In Figure 3.5(a), the bit matrix of the five data items

$$M = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

The *local bit matrix* of a data item D_j, $M(D_j)$, consists of four parts: (1) the bucket id of $B(D_j)$; (2) a subset of the bit matrix, which is formed by the bit values of the buckets within a rectangle, which includes its left and right nonempty buckets for each dimension; (3) the number of data items in bucket $B(D_j)$; (4) a distinct integer to distinguish the local bit matrix of D_j from other data items in bucket $B(D_j)$. In Figure 3.5(b), the local bit

matrix of D_3 is

$$M(D_3) = ID| \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} |1|1$$

where ID is the bucket id of $B(D_3)$.

The integrity-preserving scheme for z-dimensional data ($z > 1$) is similar as that for 1-dimensional data. Here we only show an example. Consider the five data items D_1:$(d_1^1, d_1^2), \cdots, D_5$:$(d_5^1, d_5^2)$ in Figure 3.5. The organization sends to the cloud provider the encrypted data items $(D_1)_k, \cdots, (D_5)_k$, encrypted local bit matrices $(M(D_1))_k, \cdots, (M(D_5))_k$, and the encoded data items $f_{k_1}(d_1^1), \cdots, f_{k_1}(d_5^1)$, $f_{k_2}(d_1^2), \cdots, f_{k_2}(d_5^2)$. Given a range query that includes two data items D_2 and D_3 in Figure 3.5(c), the cloud provider replies to the customer the query result $QR = \{(D_2)_k, (D_3)_k\}$ and the verification object $VO = \{(M(D_2))_k, (M(D_3))_k\}$.

Next, we analyze the detection probability for multi-dimensional data. Let $B_1, \cdots, B_m$ denote the multiple non-overlapping buckets, and $([l_i^1, h_i^1], \cdots, [l_i^z, h_i^z])$ denote a z-dimensional bucket B_i ($1 \leq i \leq m$). A bucket B_i is called a single-value bucket if for each dimension r ($1 \leq r \leq z$), $l_i^r = h_i^r$. Let $[x_1^r, x_{Nr}^r]$ denote the domain for each dimension r. Let $e(X)$ denote the frequency of the data item $X : (x^1, \cdots, x^z)$. The detection probability of a deletion operation by cloud providers can be computed as

$$Pr = \frac{\sum_{i=1}^{m} \prod_{r=1}^{z} (l_i^r - x_1^r + 1)(x_{Nr}^r - h_i^r + 1) \sum_{X \in B_i} e(X)}{\sum_{j=1}^{n} \prod_{r=1}^{z} (d_j^r - x_1^r + 1)(x_{Nr}^r - d_j^r + 1)} \tag{3.12}$$

Theorems 3 and 4 can also be extended for multi-dimensional data. We have the following two theorems.

Theorem 5. *Given any n z-dimensional data items $D_1, \cdots, D_n$, the maximum detection probability of a deletion operation is*

$$Pr_{max} = 100\% \tag{3.13}$$

if and only if each data item D_j ($1 \leq j \leq n$) forms a single-value bucket, i.e., $([d_j^1, d_j^1], \cdots, [d_j^z, d_j^z])$.

Theorem 6. *Given n z-dimensional data items $D_1, \cdots, D_n$, the minimum detection probability of a deletion operation is*

$$Pr_{min} = \frac{n}{\sum_{j=1}^{n} \prod_{r=1}^{z} (d_j^r - x_1^r + 1)(x_{N_r}^r - d_j^r + 1)} \tag{3.14}$$

if and only if there is only one bucket $([x_1^1, x_{N_1}^1], \cdots, [x_1^z, x_{N_z}^z])$.

The calculation of the detection probability in Equation 3.12 and the proofs of Theorems 5 and 6 are similar to the 1-dimensional case. Finding optimal bucket partition for multi-dimensional data is an interesting yet difficult problem and will be discussed in further work.

3.7 Evaluation

We evaluated the efficiency and effectiveness of our privacy and integrity preserving schemes for both 1-dimensional and multi-dimensional data. In terms of efficiency, we measured the data processing time for organizations, and the space cost and query processing time for cloud providers. In terms of effectiveness, we measured whether the experimental detection probability of deletion operations by cloud providers is consistent with the theoretically analysis discussed in Sections 3.5.1 and 3.6.2. Our experiments were implemented in Java 1.6.0 and carried out on a PC running Linux with 2 Intel Xeon cores and 16GB of memory.

3.7.1 Evaluation Setup

We conducted our experiments on a real data set, Adult, and five synthetic datasets. The Adult dataset is from the UCI Machine Learning Repository [32] and has been widely used in previous studies. It contains 45222 records. We chose three attributes in this dataset,

Age, Education, and Hours-per-week. Note that Education is a categorical attribute and we mapped each Education value to a distinct integer. The domains of these three attributes are $[17, 90]$, $[1, 16]$, and $[1, 99]$, respectively. The five synthetic datasets are generated by randomly choosing $10^2, 10^3, \cdots, 10^6$ data items from five domains $[0, 10^3], [0, 10^4], \cdots, [0, 10^7]$, respectively. For our order-preserving hash-based function $f_k()$, we used HMAC-MD5 with 128-bit keys as the basic hash function $h_k()$. We used the DES encryption algorithm to encrypt both data items and local bit matrices.

3.7.2 Results for 1-dimensional Data

We employed the synthetic datasets to evaluate the efficiency and effectiveness of our schemes for 1-dimensional data. For each synthetic dataset, given different number of buckets, we first computed the optimal partition and the maximum detection probability, and then we implemented our schemes using the optimal partition. We also generated 1,000 random range queries to measure the total processing time for cloud providers and verify query result integrity for customers. To process the query, we used the binary search algorithm to find out the query result. Let n denote the number of data items in a dataset and m denote the given number of buckets. According to Theorem 3, if all data items form single-value buckets, the detection probability is 100%. The rest buckets are empty buckets and the number of these buckets is $n + 1$. Thus, the total number of buckets can be computed as $2n + 1$. In other words, given $m = 2n + 1$, the output of our optimal algorithm should be these $2n + 1$ buckets. Based on this observation, we define *partition ratio* as $m/(2n + 1)$. The partition ratio helped us to normalize the results of our optimal partition algorithm for different datasets. Figure 3.6 shows the normalized results for the five synthetic datasets. We observed that the detection probability increases with the partition ratio and if partition ratio is equal to 1, *i.e.*, $m = 2n+1$, the probability becomes 1, which confirms our discussion.

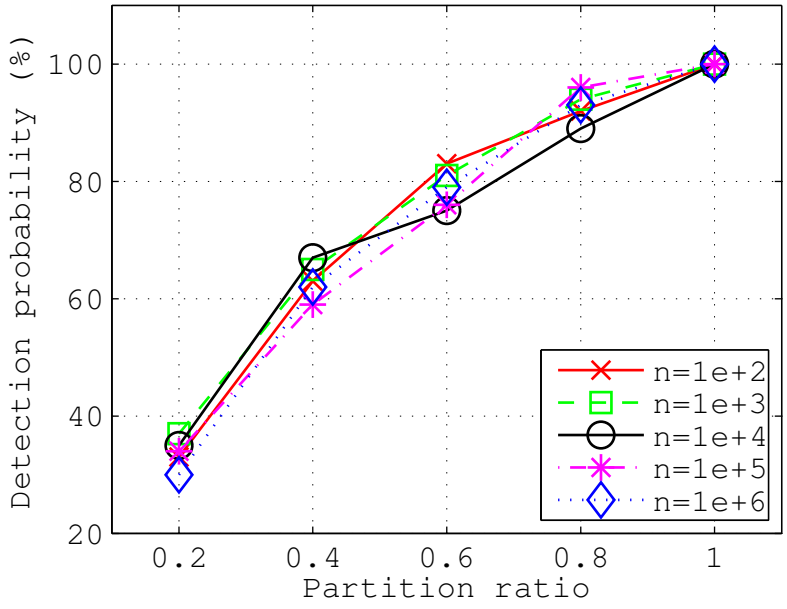


Figure 3.6. Effectiveness of optimal partition algorithm

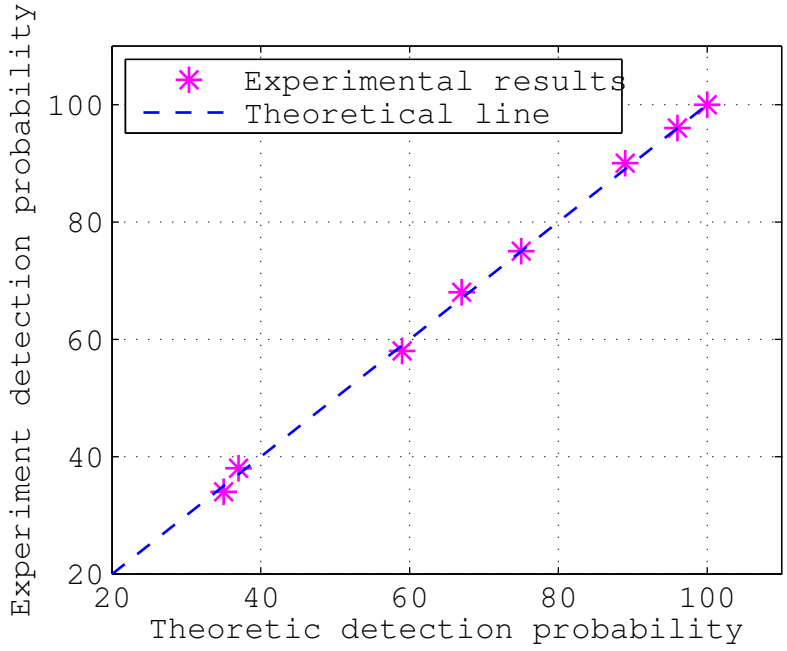


Figure 3.7. Correctness of integrity-preserving scheme

To check whether the experimental detection probability is consistent with the theoretical analysis, for each dataset, we randomly deleted a data item in each query result and then computed the percentage of query results that were detected by our integrity-preserving scheme. Note that this percentage is the experimental detection probability. Figure 3.7 shows that the experimental detection probability is close to the theoretical line, which demonstrates the correctness of our analysis.

Figures 3.8 and 3.9 show the data processing time and space cost for the five synthetic datasets, respectively. Note that the horizonal and vertical axes in these figures are in logarithmic scales. In Figure 3.8, we observed that the data processing time is less than 300 seconds for 10^5 data items. Although for one million data items, the data processing time is about 50 minutes, which is reasonable for real applications because the data processing is a one-time offline procedure. In Figure 3.9, we observed that the space cost grows linearly with the number of data items in a dataset. A cloud provider needs 33MB to store one million data items from an organization.

Figure 3.10 shows the total processing time of 1,000 queries for the five synthetic datasets. Processing 1,000 queries over one million data items only takes 2 seconds.

3.7.3 Results for Multi-dimensional Data

We employed the Adult dataset to evaluate the efficiency and effectiveness of our schemes for multi-dimensional data. The experimental results show that the data processing time for this dataset is 104 seconds, the space cost is 1.5MB, and the total processing time of 1,000 random queries is 3.5 seconds. Due to the absence of the optimal partition algorithm for multi-dimensional data, we arbitrarily partitioned the Adult dataset to different sets of buckets. The results show that the experimental detection probability is consistent with the theoretical analysis for multi-dimensional range queries.

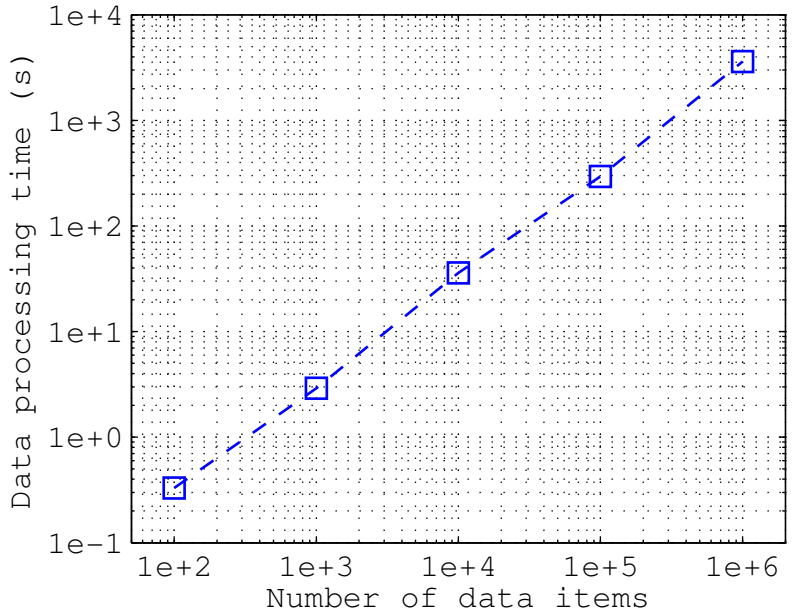


Figure 3.8. Data processing time

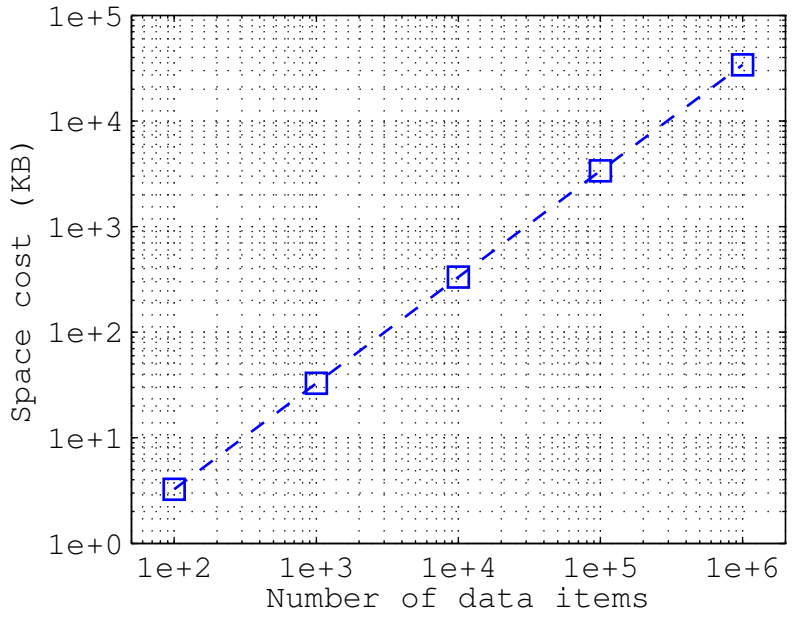


Figure 3.9. Space cost

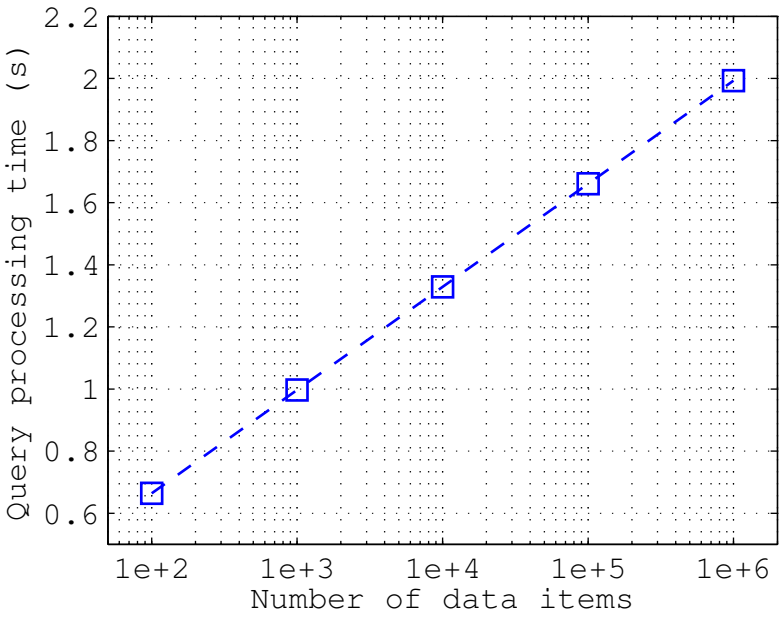


Figure 3.10. Query processing time

CHAPTER 4

Privacy Preserving Cross-Domain Cooperative Firewall Optimization

4.1 Introduction

4.1.1 Background and Motivation

Firewalls are critical in securing private networks of businesses, institutions, and home networks. A firewall is often placed at the entrance between a private network and the external network so that it can check each incoming or outgoing packet and decide whether to accept or discard the packet based on its policy. A firewall policy is usually specified as a sequence of rules, called Access Control List (ACL), and each rule has a predicate over multiple packet header fields (*i.e.*, source IP, destination IP, source port, destination port, and protocol type) and a decision (*i.e.*, accept and discard) for the packets that match the predicate. The rules in a firewall policy typically follow the first-match semantics where the decision for a packet is the decision of the first rule that the packet matches in the policy. Each physical interface of a router/firewall is configured with two ACLs: one for filtering outgoing packets and the

other one for filtering incoming packets. In this work, we use *firewalls*, *firewall policies*, and *ACLs*, interchangeably.

The number of rules in a firewall significantly affects its throughput. Figure 4.1 shows the result of the performance test of iptables conducted by HiPAC [2]. It shows that increasing the number of rules dramatically reduces the firewall throughput. Unfortunately, with explosive growth of services deployed on the Internet, firewall policies are growing rapidly in size. Thus, optimizing firewall policies is crucial for improving network performance.

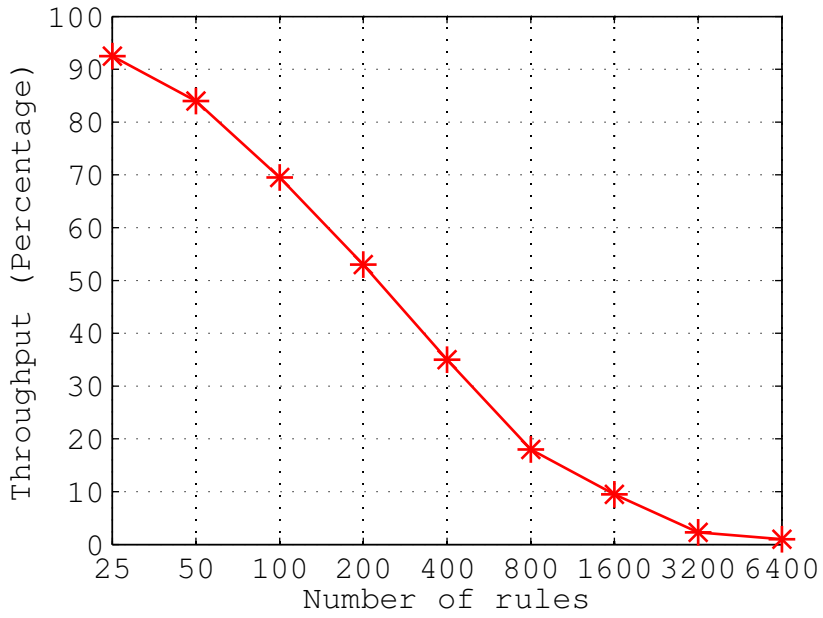


Figure 4.1. Effect of the number of rules on the throughput with frame size 128 bytes [2]

4.1.2 Limitation of Prior Work

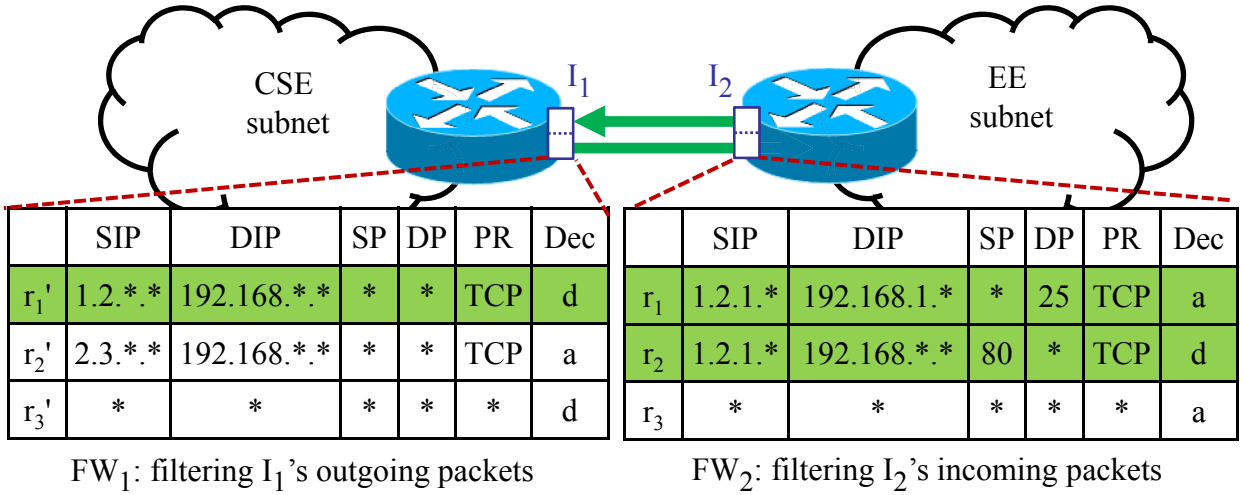
Prior work on firewall optimization focuses on either intra-firewall optimization [28, 50, 51, 52, 53, 55, 56, 57] or inter-firewall optimization [11, 81] within one administrative domain where the privacy of firewall policies is not a concern. Intra-firewall optimization means optimizing a single firewall. It is achieved by either removing redundant rules [50, 52] or rewriting rules [28, 51, 53, 55, 56, 57]. Prior work on inter-firewall optimization requires

two firewall policies without any privacy protection, and thus can only be used within one administrative domain. However, in reality, it is common that two firewalls belong to different administrative domains where firewall policies cannot be shared with each other. Keeping firewall policies confidential is important for two reasons. First, a firewall policy may have security holes that can be exploited by attackers. Quantitative studies have shown that most firewalls are misconfigured and have security holes [76]. Second, a firewall policy often contains private information, *e.g.*, the IP addresses of servers, which can be used by attackers to launch more precise and targeted attacks.

4.1.3 Cross-domain Inter-firewall Optimization

To our best knowledge, no prior work focuses on cross-domain privacy-preserving inter-firewall optimization. This work represents the first step in exploring this unknown space. Specifically, we focus on removing inter-firewall policy redundancies in a privacy-preserving manner. Consider two adjacent firewalls 1 and 2 belonging to different administrative domains Net_1 and Net_2. Let FW_1 denote the policy on firewall 1's outgoing interface to firewall 2 and FW_2 denote the policy on firewall 2's incoming interface from firewall 1. For a rule r in FW_2, if all the packets that match r but do not match any rule above r in FW_2 are discarded by FW_1, rule r can be removed because such packets never come to FW_2. We call rule r an *inter-firewall redundant rule* with respect to FW_1. Note that FW_1 and FW_2 only filter the traffic from FW_1 to FW_2; the traffic from firewall 2's outgoing interface to firewall 1's incoming interface is guarded by other two separate policies. For simplicity, we assume that FW_1 and FW_2 have no intra-firewall redundancy as such redundancy can be removed using the proposed solutions [50, 52].

Figure 4.2 illustrates inter-firewall redundancy, where two adjacent routers belong to different administrative domains CSE and EE. The physical interfaces connecting two routers



	SIP	DIP	SP	DP	PR	Dec
r_1'	1.2.*.*	192.168.*.*	*	*	TCP	d
r_2'	2.3.*.*	192.168.*.*	*	*	TCP	a
r_3'	*	*	*	*	*	d

FW_1: filtering I_1's outgoing packets

	SIP	DIP	SP	DP	PR	Dec
r_1	1.2.1.*	192.168.1.*	*	25	TCP	a
r_2	1.2.1.*	192.168.*.*	80	*	TCP	d
r_3	*	*	*	*	*	a

FW_2: filtering I_2's incoming packets

Figure 4.2. Example inter-firewall redundant rules

are denoted as I_1 and I_2, respectively. The rules of the two firewall policies FW_1 and FW_2, that are used to filter the traffic flowing from CSE to EE, are listed in two tables following the format used in Cisco Access Control Lists. Note that SIP, DIP, SP, DP, PR, and Dec denote source IP, destination IP, source port, destination port, protocol type, and decision, respectively. Clearly, all the packets that match r_1 and r_2 in FW_2 are discarded by r_1' in FW_1. Thus, r_1 and r_2 of FW_2 are inter-firewall redundant with respect to r_1' in FW_1.

4.1.4 Technical Challenges and Our Approach

The key challenge is to design a protocol that allows two adjacent firewalls to identify the inter-firewall redundancy with respect to each other without knowing the policy of the other firewall. While intra-firewall redundancy removal is complex [50, 52], inter-firewall redundancy removal with the privacy-preserving requirement is even harder. To determine whether a rule in FW_2 is inter-firewall redundant with respect to FW_1, Net_2 certainly needs some information about FW_1; yet, Net_2 cannot reveal FW_1 from such information.

A straightforward solution is to perform a privacy preserving comparison between two rules from two adjacent firewalls. Particularly, for each rule r in FW_2, this solution checks whether

all possible packets that match rule r in FW_2 match a rule r' with the discard decision in FW_1. If rule r' exists, r is inter-firewall redundant with respect to r' in FW_1. However, because firewalls follow the first-match semantics and the rules in a firewall typically overlap, this solution is not only incorrect but also incomplete. Incorrect means that wrong redundant rules could be identified in FW_2. Suppose this solution identifies r as a redundant rule in FW_2 with respect to r'_2 in FW_1. However, if some packets that match rule r also match rule r'_1 (r'_1 is above r'_2) with the accept decision in FW_1, these packets will pass through FW_1 and then FW_2 needs to filter them with r. In this case, r is actually not redundant. Incomplete means that a portion of redundant rules could be identified in FW_2. If all possible packets that match rule r in FW_2 are discarded by not only one rule but multiple rules in FW_1, r is also redundant. However, the direct comparison solution cannot identify such redundancies.

Our basic idea is as follows. For each rule r in FW_2, we first compute a set of compact predicates representing the set of packets that match r but do not match the rules above r in FW_2. Then, for each predicate, we check whether all the packets that match the predicate are discarded by FW_1. If this condition holds for all the predicates computed from rule r, then rule r is redundant. To efficiently compute these predicates, we convert firewalls to firewall decision diagrams [52]. To allow the two firewalls to detect the redundant rules in FW_2 in a privacy-preserving manner, we develop a protocol so that two firewalls can detect such redundant rules without disclosing their policies to each other.

4.1.5 Key Contributions

We make two key contributions. First, we propose a novel privacy-preserving protocol for detecting inter-firewall redundant rules in one firewall with respect to another firewall. This work represents the first effort along this unexplored direction. Second, we implemented our protocol and conducted extensive experiments on both real and synthetic firewall policies.

The results on real firewall policies show that our protocol can remove as many as 49% of the rules in a firewall whereas the average is 19.4%. The communication cost is less than a few hundred KBs. Our protocol incurs no extra online packet processing overhead and the offline processing time is less than a few hundred seconds.

4.2 System and Threat Models

4.2.1 System Model

A firewall FW is an ordered list of *rules*. Each *rule* has a *predicate* over d *fields* $F_1, \cdots, F_d$ and a *decision* for the packets that match the predicate. Firewalls usually check five fields: source IP, destination IP, source port, destination port, and protocol type. The length of these fields are 32, 32, 16, 16, and 8 bits, respectively. A *predicate* defines a set of packets over the d fields, and is specified as $F_1 \in T_1 \wedge \cdots \wedge F_d \in T_d$ where each T_i is a subset of F_i's domain $D(F_i)$. A packet over the d fields $F_1, \cdots, F_d$ is a d-tuple $(p_1, \cdots, p_d)$ where each p_i $(1 \leq i \leq d)$ is an element of $D(F_i)$. A packet $(p_1, \cdots, p_d)$ *matches* a rule $F_1 \in T_1 \wedge \cdots \wedge F_d \in T_d \rightarrow \langle decision \rangle$ if and only if the condition $p_1 \in T_1 \wedge \cdots \wedge p_d \in T_d$ holds. Typical firewall decisions include accept, discard, accept with logging, and discard with logging. Without loss of generality, we only consider accept and discard in this work. We call a rule with the accept decision an *accepting rule* and a rule with the discard decision a *discarding rule*. In a firewall policy, a packet may match multiple rules whose decisions are different. To resolve these conflicts, firewalls typically employ a first-match semantics where the decision for a packet p is the decision of the first rule that p matches. A *matching set* of r_i, $M(r_i)$, is the set of all possible packets that match the rule r_i [50]. A *resolving set* of r_i, $R(r_i)$, is the set of packets that match r_i but do not match any rule r_j above r_i $(j<i)$, and $R(r_i)$ is equal to $M(r_i) - M(r_1) \cup \cdots \cup M(r_{i-1})$ [50].

Based on above concepts, we define inter-firewall redundant rules. Given two adjacent firewalls FW_1 and FW_2, where the traffic flow is from FW_1 to FW_2, a rule r in FW_2 is inter-firewall redundant with respect to FW_1 if and only if all the packets in r's resolving set are discarded by FW_1.

4.2.2 Threat Model

We adopt the semi-honest model in [35]. For two adjacent firewalls, we assume that they are semi-honest, *i.e.*, each firewall follows our protocol correctly but each firewall may try to reveal the policy of the other firewall. The semi-honest model is realistic and well adopted [18, 78]. For example, this model is appropriate for large organizations that have many independent branches as well as for loosely connected alliances composed by multiple parties. While we are confident that all administrative domains follow mandate protocols, we may not guarantee that no corrupted employees are trying to reveal the private firewall policies of other parties. We leave investigation of privacy-preserving firewall optimization in the model with malicious participants to future work.

4.3 Privacy-Preserving Inter-Firewall Redundancy Removal

In this section, we present our privacy-preserving protocol for detecting inter-firewall redundant rules in FW_2 with respect to FW_1. To do this, we first converts each firewall to an equivalent sequence of non-overlapping rules. Because for any non-overlapping rule nr, the matching set of nr is equal to the resolving set of nr, *i.e.*, $M(nr) = R(nr)$, we only need to compare non-overlapping rules generated from the two firewalls for detecting inter-firewall redundancy. Second, we divide this problem into two subproblems, *single-rule*

coverage redundancy detection and *multi-rule coverage redundancy detection*, and then propose our privacy-preserving protocol for solving each subproblem. A rule nr is *covered* by one or multiple rules $nr'_{i_1} \cdots nr'_{i_k}$ $(k \geq 1)$ if and only if $M(nr) \subseteq M(nr'_{i_1}) \cup \cdots \cup M(nr'_{i_k})$. The first subproblem checks whether a non-overlapping rule nr in FW_2 is covered by a non-overlapping discarding rule nr' in FW_1, *i.e.*, $M(nr) \subseteq M(nr')$. The second subproblem checks whether a non-overlapping rule nr in FW_2 is covered by multiple non-overlapping discarding rules $nr'_{i_1} \cdots nr'_{i_k}$ $(k \geq 2)$ in FW_1, *i.e.*, $M(nr) \subseteq M(nr'_{i_1}) \cup \cdots \cup M(nr'_{i_k})$. Finally, after redundant non-overlapping rules generated from FW_2 are identified, we map them back to original rules in FW_2 and then identify the redundant ones.

The problem of checking whether $M(nr) \subseteq M(nr')$ boils down to the problem of checking whether one range $[a, b]$ in nr is contained by another range $[a', b']$ in nr', which further boils down to the problem of checking whether $a \in [a', b']$ and $b \in [a', b']$. Thus, we first describe the privacy-preserving protocol for comparing a number and a range.

4.3.1 Privacy-Preserving Range Comparison

To check whether a number a from FW_2 is in a range $[a', b']$ from FW_1, we use a method similar to the prefix membership verification scheme in [48]. The basic idea is to convert the problem of checking whether $a \in [a', b']$ to the problem of checking whether two sets converted from a and $[a', b']$ have a common element. Our method consists of four steps:

(1) *Prefix conversion.* It converts $[a', b']$ to a minimum number of prefixes, denoted as $\mathcal{T}([a', b'])$, whose union is $[a', b']$. For example, $\mathcal{T}([11, 15])$=$\{1011, 11^{**}\}$.

(2) *Prefix family construction.* It generates all the prefixes that contains a including a itself. This set of prefixes is called the prefix family of a, denoted as $\mathcal{F}(a)$. Let k be the bit length of a. The prefix family $\mathcal{F}(a)$ consists of $k + 1$ prefixes where the i-th prefix is obtained by replacing the last $i - 1$ bits of a by $*$. For example, as the binary representation

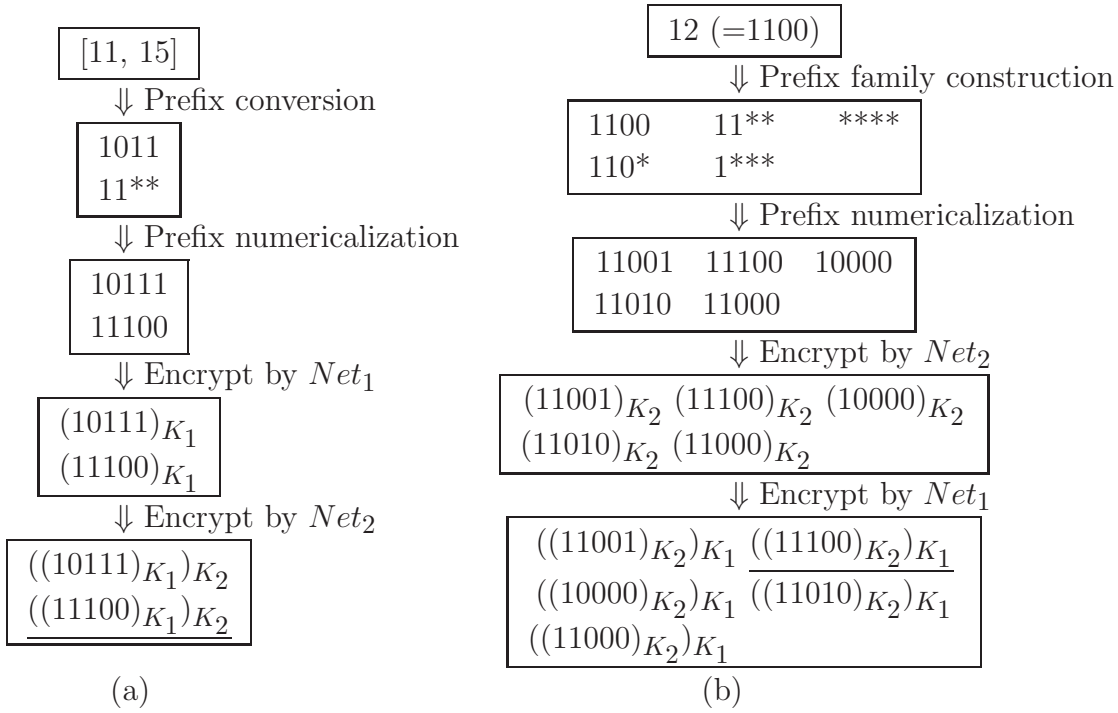


Figure 4.3. Prefix membership verification

of 12 is 1100, we have $\mathcal{F}(12)$={1100, 110*, 11**, 1***, ****}. It is not difficult to prove that $a \in [a', b']$ if and only if $\mathcal{F}(a) \cap \mathcal{T}([a', b']) \neq \emptyset$.

(3) *Prefix numericalization*. It converts the prefixes generated in the previous steps to concrete numbers such that we can encrypt them in the next step. We use the prefix numericalization scheme in [20]. Given a prefix $b_1 b_2 \cdots b_k {*}\cdots{*}$ of w bits, we first insert 1 after b_k. The bit 1 represents a separator between $b_1 b_2 \cdots b_k$ and $*\cdots*$. Then we replace every $*$ by 0. For example, 11** is converted to 11100. If the prefix does not contain *s, we place 1 at the end. For example, 1100 is converted to 11001.

(4) *Comparison*. It checks whether $a \in [a', b']$ by checking whether $\mathcal{F}(a) \cap \mathcal{T}([a', b']) \neq \emptyset$, which boils down to checking whether two numbers are equal. We use commutative encryption to do this checking in a privacy-preserving manner. Given a number x and two encryption keys K_1 and K_2, a commutative encryption is a function that satisfies the property $((x)_{K_1})_{K_2} = ((x)_{K_2})_{K_1}$, *i.e.*, encryption with key K_1 first and then K_2 is equivalent

to encryption with key K_2 first and then K_1. Example commutative encryption algorithms are the Pohlig-Hellman Exponentiation Cipher [64] and Secure RPC Authentication (SRA) [67]. Each domain chooses a private key. Let K_1, K_2 be the private keys chosen by Net_1 and Net_2, respectively. To check whether number v_1 from Net_1 is equal to number v_2 from Net_2 without disclosing the value of each number to the other party, Net_1 can first encrypt v_1 using key K_1 and sends $(x)_{K_1}$ to Net_2; similarly, Net_2 can first encrypt v_2 using key K_2 and sends $(x)_{K_2}$ to Net_1. Then, each party checks whether $v_1 = v_2$ by checking whether $((v_1)_{K_1})_{K_2} = ((v_2)_{K_2})_{K_1}$. Note that $((v_1)_{K_1})_{K_2} =((v_2)_{K_2})_{K_1}$ if and only if $v_1 = v_2$. If $v_1 \neq v_2$, neither party can learn anything about the numbers being compared. Figure 4.3 illustrates the process of checking whether 12 from FW_2 is in the range $[11, 15]$ from FW_1.

4.3.2 Processing Firewall FW_1

To detect redundant rules in FW_2, Net_1 converts its firewall FW_1 to a set of non-overlapping rules. To preserve the privacy of FW_1, Net_1 first converts each range of a non-overlapping discarding rules from FW_1 to a set of prefixes. Second, Net_1 and Net_2 encrypt these prefixes using commutative encryption. The conversion of FW_1 includes nine steps:

(1) Net_1 first converts FW_1 to an equivalent firewall decision diagram (FDD) [36, 37]. An FDD for a firewall FW of a sequence of rules $\langle r_1, \cdots, r_n \rangle$ over fields $F_1, \cdots, F_d$ is an acyclic and directed graph that has five properties. (a) There is exactly one node that has no incoming edges. This node is called the *root*. The nodes that have no outgoing edge are called *terminal* nodes. (b) Each node v has a label, denoted $F(v)$. If v is a nonterminal node, then $F(v) \in \{F_1, \cdots, F_d\}$. If v is a terminal node, then $F(v)$ is a decision. (c) Each edge e, $u \rightarrow v$, is labeled with a non-empty set of integers, denoted $I(e)$, where $I(e)$ is a subset of the domain of u's label (*i.e.*, $I(e) \subseteq D(F(u))$). (d) The set of all outgoing edges of a node v, denoted $E(v)$, satisfies two conditions: (a) *consistency*: $I(e) \cap I(e') = \emptyset$ for any two distinct

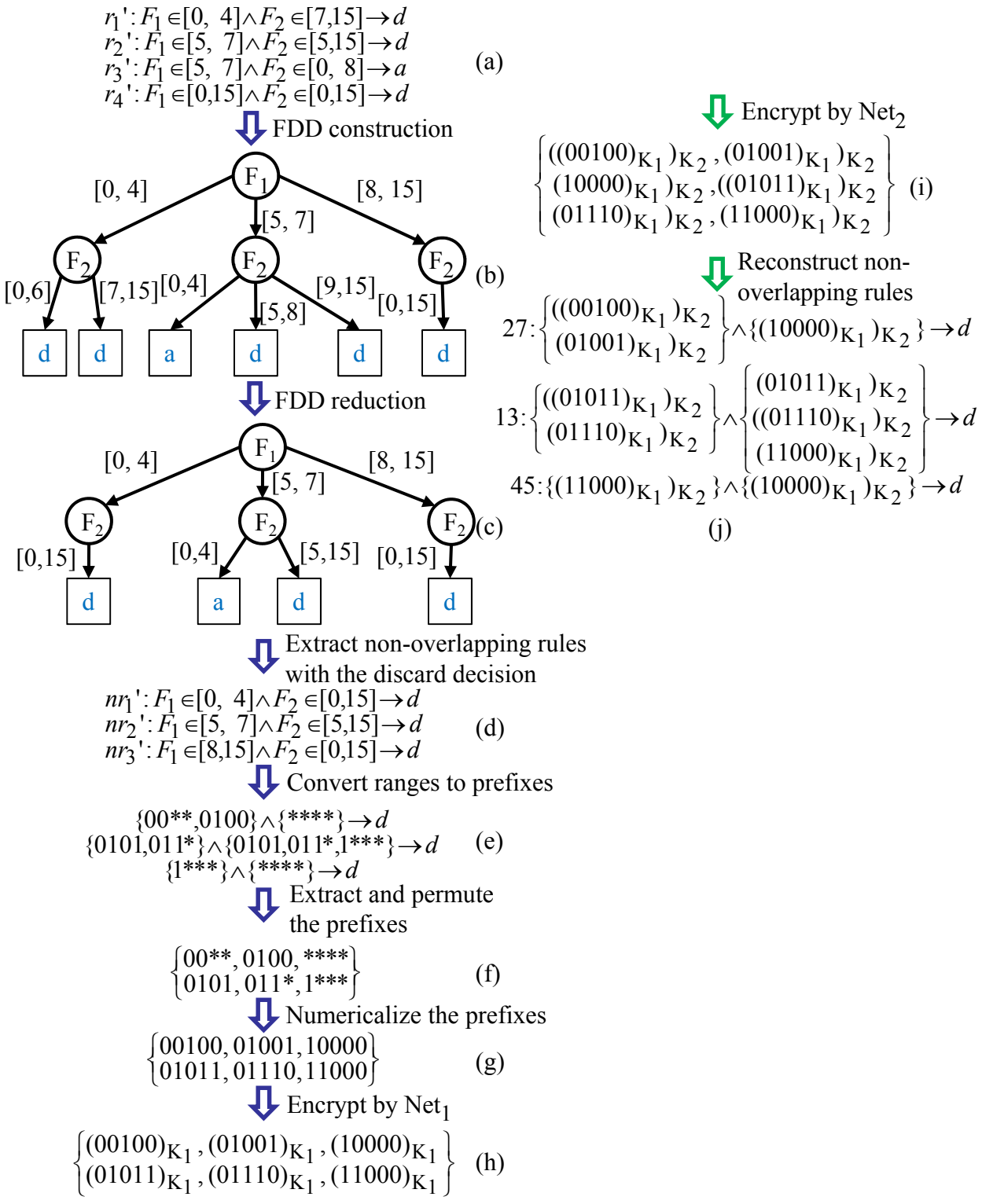


Figure 4.4. The Conversion of FW_1

edges e and e' in $E(v)$; (b) *completeness*: $\bigcup_{e \in E(v)} I(e) = D(F(v))$. (e) A directed path from the root to a terminal node is called a *decision path*. No two nodes on a decision path have the same label. Each path in the FDD corresponds to a non-overlapping rule. A *full-length ordered FDD* is an FDD where in each decision path all fields appear exactly once and in the same order. For ease of presentation, we use the term "FDD" to denote "full-length ordered FDD". An FDD construction algorithm, which converts a firewall policy to an equivalent FDD, is presented in [49]. Figure 4.4(b) shows the FDD constructed from Figure 4.4(a).

(2) Net_1 reduces the FDD's size by merging isomorphic subgraphs. An FDD f is *reduced* if and only if it satisfies two conditions: (a) no two nodes in f are isomorphic; (b) no two nodes have more than one edge between them. Two nodes v and v' in an FDD are *isomorphic* if and only if v and v' satisfy one of the following two conditions: (a) both v and v' are terminal nodes with identical labels; (b) both v and v' are nonterminal nodes and there is a one-to-one correspondence between the outgoing edges of v and the outgoing edges of v' such that every two corresponding edges have identical labels and they both point to the same node. Figure 4.4(c) shows the FDD reduced from the FDD in Figure 4.4(b).

(3) Net_1 extracts non-overlapping discarding rules. Net_1 does not extract non-overlapping accepting rules because the packets accepted by FW_1 are passed to FW_2. Note that a non-overlapping rule from FW_2 that is covered by these discarding rules from FW_1 is redundant. Figure 4.4(d) shows the discarding non-overlapping rules extracted from the reduced FDD in Figure 4.4(c).

(4) Net_1 converts each range to a set of prefixes. Figure 4.4(e) shows the prefixes generated from Figure 4.4(d).

(5) Net_1 unions all these prefix sets and then permutes the prefixes. Figure 4.4(f) shows the resulting prefix set. Note that the resulting set does not include duplicate prefixes. The benefits are two-fold. In terms of efficiency, it avoids encrypting and sending duplicate prefixes for both parties, and hence, significantly reduces computation and communication

costs. In terms of security, Net_2 cannot reconstruct the non-overlapping rules from FW_1, because Net_2 does not know which prefix belongs to which field of which rule. However, Net_1 knows such information and it can reconstruct these non-overlapping rules.

(6) Net_1 numericalizes and encrypts each prefix using K_1, and then sends to Net_2. Figures 4.4(g) and (h) show the numericalized and encrypted prefixes, respectively.

(7) Net_2 further encrypts these prefixes with K_2 and sends them back to Net_1 as shown in Figure 4.4(i).

(8) Net_1 reconstructs its non-overlapping discarding rules from the double encrypted prefixes because Net_1 knows which prefix belongs to which field of which rule.

(9) For each reconstructed non-overlapping rule, Net_1 assigns a distinct random index to it. These indices are used for Net_2 to identify the redundant non-overlapping rules from FW_2. For example, in Figure 4.6(a), Net_1 assigns its three rules with three random indices: 27, 13, and 45. The detailed discussion is in Section 4.3.4.

4.3.3 Processing Firewall FW_2

In order to compare two firewalls in a privacy-preserving manner, Net_2 and Net_1 convert firewall FW_2 to d sets of double encrypted numbers, where d is the number of fields. The conversion of FW_2 includes five steps:

(1) Net_2 converts FW_2 to an equivalent all-match FDD. All-match FDDs differ from FDDs on terminal nodes. In an all-match FDD, each terminal node is labeled with a nonempty set of rule sequence numbers, whereas in an FDD each terminal node is labeled with a decision. For rule r_i ($1 \leq i \leq n$), we call i the *sequence number* of r_i. The set of rule sequence numbers labeled on a terminal node consists of the sequence numbers of all the rules that overlaps with the decision path ending with this terminal node. Given a decision path $\mathcal{P}$, $(v_1 e_1 \cdots v_d e_d v_{d+1})$, the matching set of $\mathcal{P}$ is defined as the set of all packets that satisfy

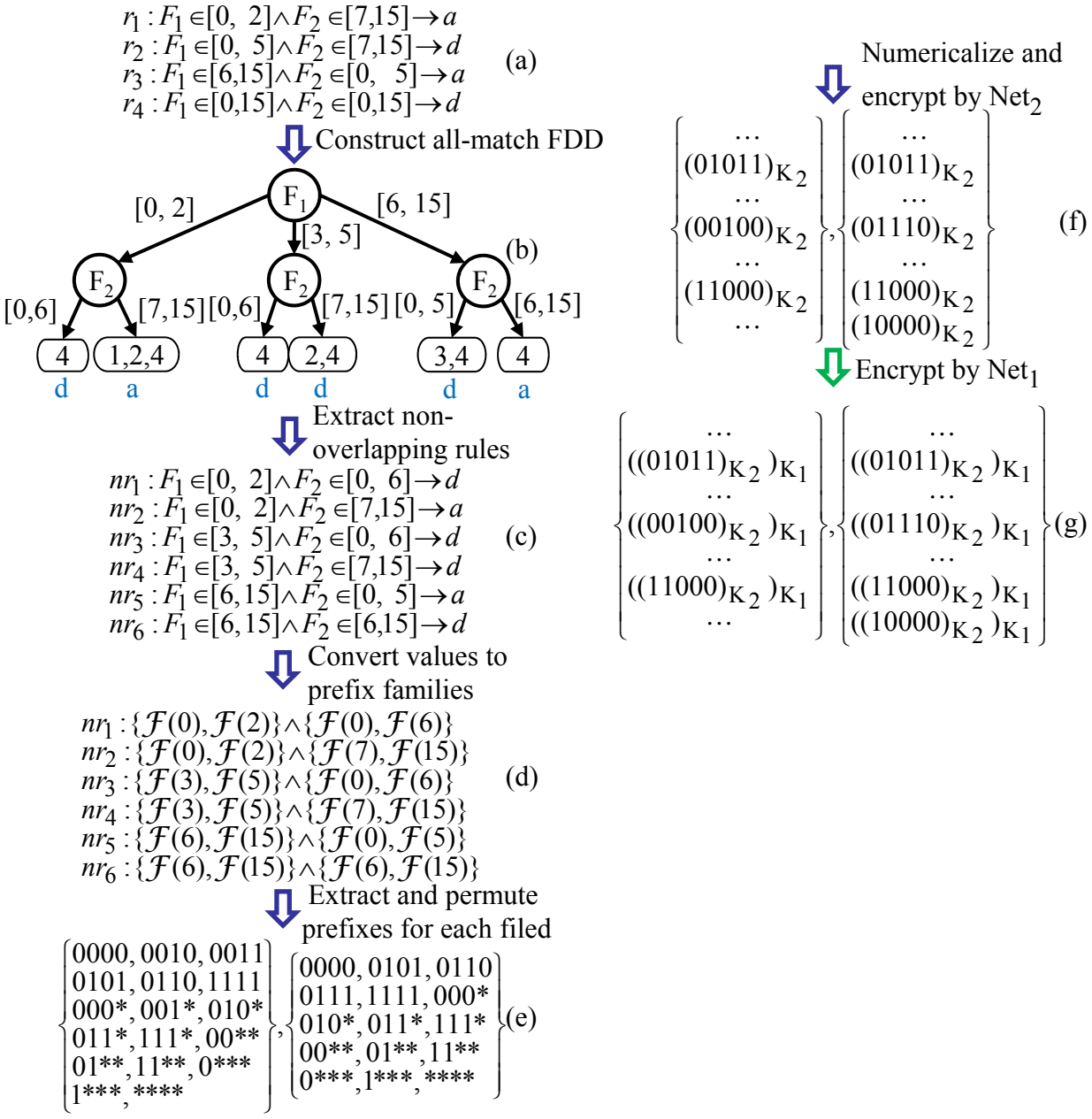


Figure 4.5. The Conversion of FW_2

$F(v_1) \in I(e_1) \wedge \cdots \wedge F(v_d) \in I(e_d)$. We use $M(\mathcal{P})$ to denote the matching set of $\mathcal{P}$. More formally, in an all-match FDD, for any decision path $\mathcal{P}$: $(v_1 e_1 \cdots v_d e_d v_{d+1})$, if $M(\mathcal{P}) \cap M(r_i) \neq \emptyset$, then $M(\mathcal{P}) \subseteq M(r_i)$ and $i \in F(v_{d+1})$. Figure 4.4(b) shows the all-match FDD generated from Figure 4.4(a). Considering the terminal node of the fourth path in Figure 4.4(b), its label {2, 4} means that $M(\mathcal{P}) \subseteq M(r_2)$, $M(\mathcal{P}) \subseteq M(r_4)$, $M(\mathcal{P}) \cap M(r_1) = \emptyset$, and $M(\mathcal{P}) \cap M(r_3) = \emptyset$. An all-match FDD not only represents a firewall in a non-overlapping fashion but also represents the overlapping relationship among rules. The reason of converting FW_2 to an all-match FDD is that later Net_2 needs the rule sequence numbers to identify inter-firewall redundant rules in FW_2.

(2) Net_2 extracts all non-overlapping rules from the all-match FDD. Figure 4.5(c) shows the non-overlapping rules extracted from Figure 4.5(b). For each range $[a, b]$ of a non-overlapping rule, Net_2 generates two prefix families $\mathcal{F}(a)$ and $\mathcal{F}(b)$. Figure 4.5(d) shows the result from Figure 4.5(c).

(3) For every field F_k, Net_2 unions all prefix families of all the non-overlapping rules into one prefix set and permutes the prefixes. Considering the first field F_1 in Figure 4.5(d), Net_2 unions $\mathcal{F}(0)$, $\mathcal{F}(2)$, $\mathcal{F}(3)$, $\mathcal{F}(5)$, $\mathcal{F}(6)$ and $\mathcal{F}(15)$ to the first prefix set in Figure 4.5(e). The benefits of this step are similar to those of Step (5) in FW_1's conversion. In terms of efficiency, it avoids encrypting and sending the duplicate prefixes for each field, and hence, significantly reduces the computation and communication costs. In terms of security, Net_1 cannot reconstruct the non-overlapping rules of FW_2, because Net_1 does not know which prefix belongs to which rule in FW_2. However, Net_2 knows such information, which will be used to identify redundant non-overlapping rules later. Note that, the ordering of the fields cannot be permuted because Net_1 needs to perform comparison of the prefixes with only those prefixes from the corresponding fields.

(4) Net_2 numericalizes and encrypts the prefixes using its private K_2, and sends them to Net_1. Figure 4.5(f) shows the prefixes.

(5) Net_1 further encrypts these prefixes using key K_1.

4.3.4 Single-Rule Coverage Redundancy Detection

After processing the two firewalls, Net_1 has a sequence of double encrypted non-overlapping rules obtained from FW_1 and d sets of double encrypted numbers obtained from FW_2. Let $(F_1 \in \mathcal{T}_1) \wedge \cdots \wedge (F_d \in \mathcal{T}_d) \rightarrow discard$ denote a double encrypted rule, where $\mathcal{T}_i$ is a set of double encrypted numbers. Let $\mathbb{T}_1, \cdots, \mathbb{T}_d$ denote the d sets of double encrypted numbers from FW_2. Figure 4.6(a) shows the double encrypted non-overlapping rules generated from Figure 4.4 and Figure 4.6(b) shows the double encrypted numbers generated from Figure 4.5. For each field F_i $(1 \leq i \leq d)$ and for each number a in $\mathbb{T}_i$, Net_1 checks whether there exists a double encrypted rule $(F_1 \in \mathcal{T}_1) \wedge \cdots \wedge (F_d \in \mathcal{T}_d) \rightarrow discard$ such that $a \in \mathcal{T}_i$. If rule r_i satisfies this condition, then Net_1 associates the rule index i with a. As there maybe multiple rules that satisfy this condition, eventually Net_1 associates a set of rule indices with a. If no rule satisfies this condition, Net_1 associates an empty set with a. Considering the number $((01011)_{K_2})_{K_1}$, only the rule with index 13 contains it because $((01011)_{K_2})_{K_1} = ((01011)_{K_1})_{K_2}$; thus, Net_1 associates $((01011)_{K_2})_{K_1}$ with $\{13\}$. Finally, Net_1 replaces each number in $\mathbb{T}_1, \cdots, \mathbb{T}_d$ with its corresponding set of rule indices, and sends them to Net_2.

Upon receiving the sets from Net_1, for each prefix family, Net_2 finds the index of the rule that overlaps with the prefix family. For a non-overlapping rule nr from FW_2, if all its prefix families overlap with the same discarding rule nr' from FW_1, nr is covered by nr' and hence, nr is redundant. For example, in Figure 4.6(d), nr_1 is redundant, because $\mathcal{F}(0)$, $\mathcal{F}(2)$, $\mathcal{F}(0)$ and $\mathcal{F}(6)$ overlap with rule 27 from FW_1. Similarly, nr_2 is redundant. Note that $\mathcal{F}(v)$:$j_1, ..., j_k$ denotes that $\mathcal{F}(v)$ overlaps with non-overlapping rules $j_1, ..., j_k$ from FW_1.

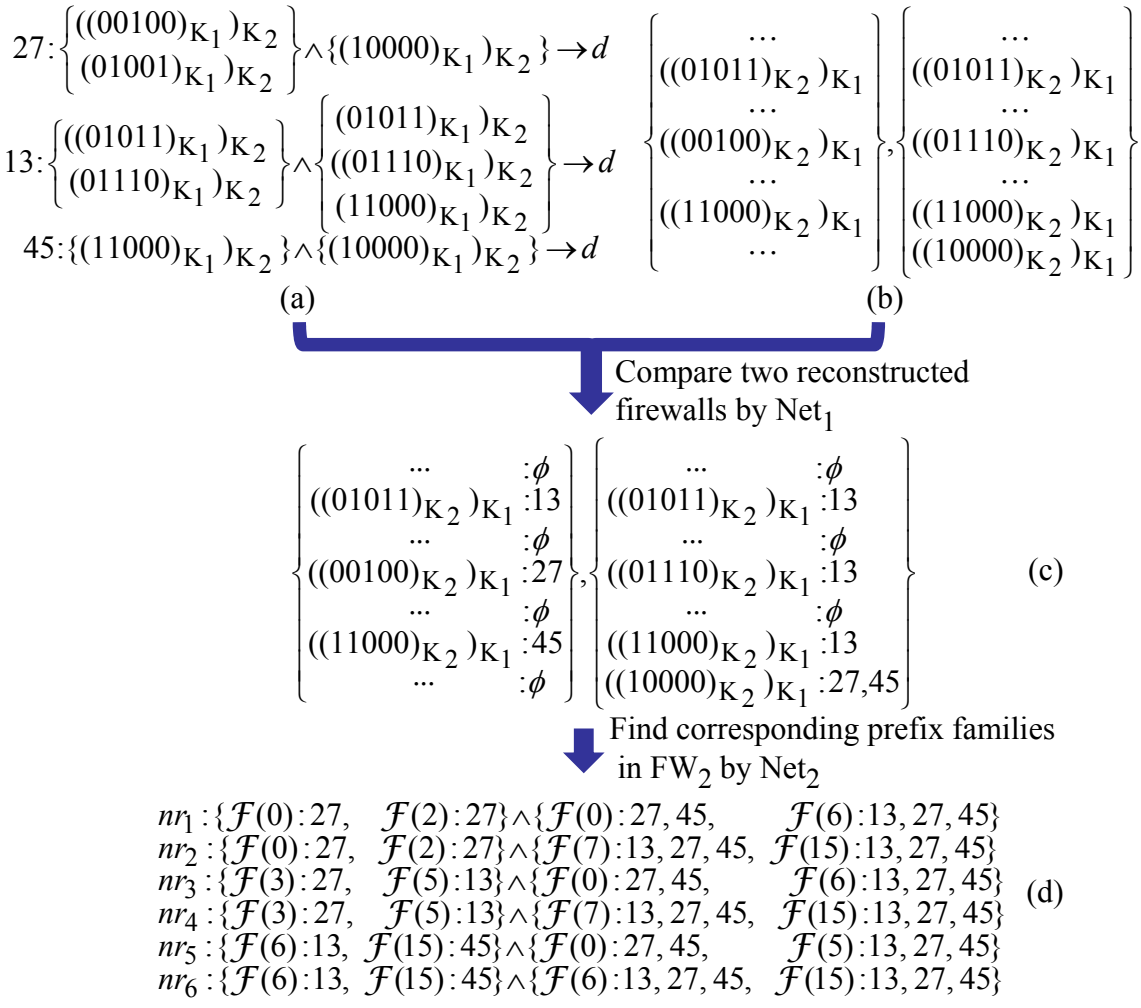


Figure 4.6. Comparison of Two Firewalls

4.3.5 Multi-Rule Coverage Redundancy Detection

To detect multi-rule coverage redundancy, our basic idea is to combine all the non-overlapping discarding rules from FW_1 to a set of new rules such that for any arbitrary rule from FW_2, if it is covered by multiple non-overlapping discarding rules from FW_1, it is covered by a rule from these new rules. More formally, let $nr'_1, \cdots, nr'_l$ denote the non-overlapping discarding rules from FW_1 and $s'_1, \cdots, s'_g$ denote the set of new rules generated from $nr'_1, \cdots, nr'_l$. For any rule nr from FW_2, if a non-overlapping rule nr from FW_2 is multi-rule coverage redundant, *i.e.*, $M(nr) \subseteq M(nr'_{i_1}) \cup \cdots \cup M(nr'_{i_k})$ where $nr'_{i_1} \cdots nr'_{i_k}$ $(k \geq 2)$ from FW_1, there is a rule s'_j $(1 \leq j \leq g)$ that cover nr, *i.e.*, $M(nr) \subseteq M(s'_j)$. Thus, after Net_1 computes the set of new rules from FW_1, we reduce the problem of multi-rule coverage redundancy detection to single-rule coverage redundancy detection. Then, two parties Net_1 and Net_2 can cooperatively run our protocol proposed in this section to identify the non-overlapping single-rule and multi-rule coverage redundant rules from FW_2 at the same time. However, the key question is how to compute the set of new rules $s'_1, \cdots, s'_g$.

A straightforward method is to compute all possible rules that are covered by a single or multiple non-overlapping discarding rules among $nr'_1, \cdots, nr'_l$. All these rules form the set of new rules $s'_1, \cdots, s'_g$. However, this method incurs two major drawbacks. First, the time and space complexities of this method can be significant because the number of all these rules could be huge. Second, due to the huge number of these rules, the communication and computation costs increase significantly. The relationship between these costs and the number of rules is discussed in Section 4.5.2.

Our solution is to compute only the *largest* rules that are covered by a single or multiple non-overlapping discarding rules among $nr'_1, \cdots, nr'_l$. The term *largest* can be explained as follows. Without considering the decision, a firewall rule with d fields can be denoted as a hyperrectangle over a d-dimensional space. Then, l non-overlapping discarding rules

$nr'_1, \cdots, nr'_l$ are l hyperrectangles over a d-dimensional space. The new rules $s'_1, \cdots, s'_g$ are also the hyperrectangles. The term *largest* means that if a hyperrectangle s^*_j contains the hyperrectangle s'_j but is larger than s'_j, s^*_j has some parts which are not covered by all the l hyperrectangles $nr'_1, \cdots, nr'_l$. For example, the non-overlapping discarding rules nr'_1, nr'_2, nr'_3 in Figure 4.4(d) can be illustrated as three filled rectangles in Figure 4.7. Figure 4.7 also shows three new rules s'_1, s'_2, s'_3 generated from nr'_1, nr'_2, nr'_3, where s'_1 is illustrated in the dashed rectangle, s'_2 is the same as nr'_2, and s'_3 is the same as nr'_3. Note that the values of two fields F_1 and F_2 are integers. Thus, we can combine three ranges [0, 4], [5, 7], and [8, 15] to a range [0, 15] for the field F_1 of s'_1.

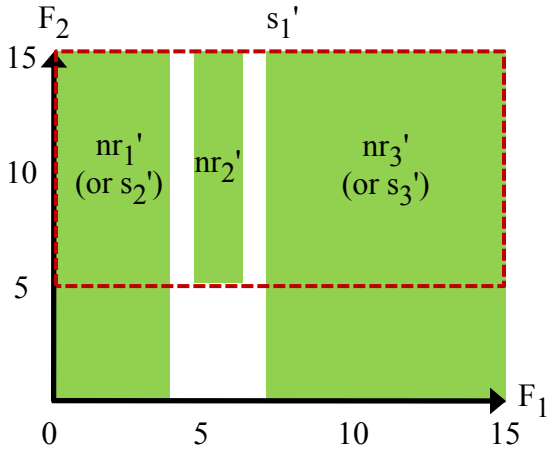


Figure 4.7. Three largest rules generated from Figure 4.4(d)

More formally, we can define a largest rule s'_j $(1 \leq j \leq g)$ as follows.

1. $M(s'_j) \subseteq M(nr'_1) \cup \cdots \cup M(nr'_l)$.

2. For any rule s^*_j that $M(s^*_j) \supset M(s'_j)$,
   $M(s^*_j) \not\subset M(nr'_1) \cup \cdots \cup M(nr'_l)$.

Given the set of all largest rules $s'_1, \cdots, s'_g$ generated from $nr'_1, \cdots, nr'_l$, for any rule nr, if nr is covered by one or multiple rules in $nr'_1, \cdots, nr'_l$, there exists a largest rule s'_j $(1 \leq j \leq g)$ which covers nr. We have the following theorem.

Theorem 7. *Given the set of all largest rules $s'_1, \cdots, s'_g$ generated from $nr'_1, \cdots, nr'_l$, for any rule nr, if $M(nr) \subseteq M(nr'_1) \cup \cdots \cup M(nr'_l)$, there exists a largest rule s'_j $(1 \leq j \leq g)$ which satisfies the condition $M(nr) \subseteq M(s'_j)$.*

Proof. If nr is a largest rule, it is included in $s'_1, \cdots, s'_g$. If nr is not a largest rule, we prove it by contradiction. If there exists no largest rule among $s'_1, \cdots, s'_g$ which covers nr, we can generate all possible rules which satisfy two conditions: (1) the matching set of each of these rules is the superset of $M(nr)$; (2) each of these rules is covered by $nr'_1, \cdots, nr'_l$. Among all these generated rules, there exists at least a largest rule otherwise the number of these rules is infinite. However, $M(nr'_1) \cup \cdots \cup M(nr'_l)$ is a finite domain. □

Next, we discuss how to compute the set of all the largest rules $S = \{s'_1, \cdots, s'_g\}$ from the non-overlapping discarding rules $nr'_1, \cdots, nr'_l$. Our idea is to first compute the largest rules for every two rules among $nr'_1, \cdots, nr'_l$. Repeat computing the largest rules for every two rules in the previous step until the resulting rules do not change. Finally, the resulting rules is the set of all the largest rules $s'_1, \cdots, s'_g$. Note that it is trivial to compute the largest rules from two rules. This algorithm is shown in Algorithm 2.

For example, to identify the single-rule and multiple-rule coverage redundancy simultaneously, Net_1 only needs to perform one more step between Figure 4.4(d) and (e). Net_1 computes all the largest rules from the non-overlapping discarding rules in Figure 4.4(d), which are

$$s'_1 : F_1 \in [0, 15] \wedge F_2 \in [5, 15] \rightarrow d$$

$$s'_2 : F_1 \in [0, 4] \wedge F_2 \in [0, 15] \rightarrow d$$

$$s'_3 : F_1 \in [8, 15] \wedge F_2 \in [0, 15] \rightarrow d$$

Finally, Net_2 can identify that $nr_4 : F_1 \in [3, 15] \wedge F_2 \in [7, 15] \rightarrow d$ is redundant because nr_4 is covered the by the rule s'_1.

Algorithm 2: Computation of the set of largest rules

Input: l non-overlapping rules $nr'_1, \cdots, nr'_l$.

Output: The set of all the largest rules S

```
1 Initialize S to {nr'_1, ..., nr'_l};
2 while S has been changed do
3     for every two rules s'_i, s'_j (i ≠ j) in S do
4         remove s'_i and s'_j from S;
5         compute the largest rules from s'_i and s'_j;
6         add the largest rules to S';
7     S = S';
8 return S;
```

4.3.6 Identification and Removal of Redundant Rules

After single-rule and multi-rule coverage redundancy detection, Net_2 identifies the redundant non-overlapping rules in FW_2. Next, Net_2 needs to identify which original rules are inter-firewall redundant. As each path in the all-match FDD of FW_2 corresponds to a non-overlapping rule, we call the paths that correspond to the redundant non-overlapping rules *redundant paths* and the remaining paths *effective paths*. For example, in Figure 4.8, the dashed paths are the redundant paths that correspond to nr_1, nr_2 and nr_4 in Figure 4.5(c), respectively. Finally, Net_2 identifies redundant rules based on Theorem 8.

Theorem 8. *Given firewall FW_2:$\langle r_1, \cdots, r_n \rangle$ with no intra-firewall redundancy and its all-match FDD, rule r_i is inter-firewall redundant with respect to FW_1 if and only if two conditions hold: (1) there is a redundant path whose terminal node contains sequence number i; (2) there is no effective path whose terminal node contains i as the smallest element.*

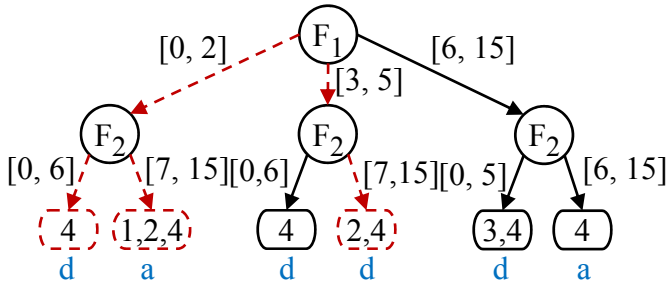


Figure 4.8. Identification of redundant rules in FW_2

Proof. Let $\{\mathcal{P}_1,\cdots,\mathcal{P}_m\}$ denote all paths in FW_2's all-match FDD. According to the theorems in [50, 52], the resolving set of each rule r_i $(1\leq i\leq n)$ in firewall FW_2 satisfies the condition $R(r_i)=\cup_{k=1}^{k=t}M(\mathcal{P}_{j_k})$ $(1\leq j_k\leq m)$, where $\mathcal{P}_{j_1},\cdots,\mathcal{P}_{j_t}$ are all the paths whose terminal nodes contain i as the smallest element. Based on the definition of inter-firewall redundant rules in Section 4.2.1, rule r_i is inter-firewall redundant if and only if all the packets in $\cup_{k=1}^{k=t}M(\mathcal{P}_{j_k})$ are discarded by FW_1. Thus, each path $\mathcal{P}_{j_k}$ $(1\leq k\leq t)$ is a redundant path. In other words, all the paths $\mathcal{P}_{j_1},\cdots,\mathcal{P}_{j_t}$ whose terminal nodes contain i as the smallest element are redundant paths. □

Considering redundant paths in Figure 4.8, Net_2 identifies that r_1 and r_2 are inter-firewall redundant with respect to FW_1.

Theorem 9. *The privacy-preserving inter-firewall redundancy removal protocol is a* complete *inter-firewall redundancy removal scheme.*

Proof. Suppose that our proposed protocol is not a complete scheme and hence it cannot detect all inter-firewall redundant rules in FW_2. Assume that rule r_i is inter-firewall redundant in FW_2 but it is not detected by our protocol. According to Theorem 8, $R(r_i)=\cup_{k=1}^{k=t}M(\mathcal{P}_{j_k})$ $(1\leq j_k\leq m)$ and $\mathcal{P}_{j_1},\cdots,\mathcal{P}_{j_t}$ are redundant paths in FW_2's all-match FDD. Thus, some paths in $\{\mathcal{P}_{j_1},\cdots,\mathcal{P}_{j_t}\}$ cannot be identified as redundant paths by our protocol. This conclusion violates the fact that our protocol can identify all redundant paths in FW_2's FDD. □

4.4 Firewall Update After Optimization

If FW_1 or FW_2 changes after inter-firewall optimization, the inter-firewall redundant rules identified by the optimization may not be inter-firewall redundant anymore. In this section, we discuss our solution to address firewall update. There are five possible cases.

(1) Net_1 changes the decisions of some rules from discard to accept in FW_1. In this case, Net_1 needs to notify Net_2 that which non-overlapping rules (indices of these rules) from FW_1 are changed. Using this information, Net_2 checks if there were any rules in FW_2 that were removed due to these rules, and then adds the affected rules back into FW_2.

(2) Net_1 changes the decisions of some rules from accept to discard in FW_1. In this case, Net_2 can run our cooperative optimization protocol again to identify more inter-firewall redundant rules in FW_2.

(3) Net_2 changes the decisions of some rules in FW_2. In this case, neither party needs to take actions because the inter-firewall redundancy detection does not consider the decisions of the rules in FW_2.

(4) Net_1 adds or removes some rules in FW_1. In this case, since the resolving sets of some rules in FW_1 may change, a rule in FW_2 that used to be inter-firewall redundant maybe not redundant anymore. It is important for Net_2 to run our optimization protocol again.

(5) Net_2 adds or removes some rules in FW_2. Similar to the fourth case, since the resolving sets of some rules in FW_2 may change, it is important for Net_2 to run our protocol again.

4.5 Security and Complexity Analysis

4.5.1 Security Analysis

To analyze the security of our protocol, we first describe the commutative encryption and its properties. Let Key denote a set of private keys and Dom denote a finite domain. A

commutative encryption f is a computable function f:$Key \times Dom \rightarrow Dom$ that satisfies the following four properties. (1) Secrecy: For any x and key K, given $(x)_K$, it is computationally infeasible to compute K. (2) Commutativity: For any x, K_1, and K_2, we have $((x)_{K_1})_{K_2} = ((x)_{K_2})_{K_1}$. (3) For any x, y, and K, if $x \neq y$, we have $(x)_K \neq (y)_K$. (4) The distribution of $(x)_K$ is indistinguishable from the distribution of x. Note that, for ease of presentation, in the rest of this section, any x, y, or z is an element of Dom, any K, K_1, or K_2 is an element of Key, and $(x)_K$ denotes $f(x, K)$.

In the conversion of FW_1, for each non-overlapping rule nr' from FW_1, let $V_{F_j}(nr')$ denote the prefix set for the field F_j, *e.g.*, in Figure 4.4(e), $V_{F_1}(nr'_1)$ denotes {00**, 0100}. In the conversion of FW_2, let U_{F_j} denote the prefix set for the field F_j after removing duplicate prefixes, *e.g.*, in Figure 4.5(e), U_{F_1} denotes the first prefix set. Our cooperative optimization protocol essentially compares $V_{F_j}(nr')$ and U_{F_j} in a privacy-preserving manner. We have the following theorem.

Theorem 10. *If both parties Net_1 and Net_2 are semi-honest, after comparing two sets $V_{F_j}(nr')$ and U_{F_j} using our protocol, Net_1 learns only the size $|U_{F_j}|$ and the intersection $V_{F_j}(nr') \cap U_{F_j}$, and Net_2 learns only the size $|V_{F_j}(nr')|$ and the intersection $V_{F_j}(nr') \cap U_{F_j}$.*

Proof. According to the theorems in multi-party secure computation [9, 34], if we can prove that the distribution of the Net_1's view of our protocol cannot be distinguished from a simulation that uses only $V_{F_j}(nr')$, $V_{F_j}(nr') \cap U_{F_j}$, and $|U_{F_j}|$, then Net_1 cannot learn anything else except $V_{F_j}(nr') \cap U_{F_j}$ and $|U_{F_j}|$. Note that Net_1's view of our protocol is the information that Net_1 gains from FW_2.

Without loss of generality, we only prove that Net_1 learns only the size $|U_{F_j}|$ and the intersection $V_{F_j}(nr') \cap U_{F_j}$. The simulator for Net_1 uses key K_1 to create a set from $V_{F_j}(nr')$ and $V_{F_j}(nr') \cap U_{F_j}$ as follows

$$Y_S = \{\underbrace{(x_1)_{K_1}, \cdots, (x_m)_{K_1}}_{x_i \in V_{F_j}(nr') \cap U_{F_j}}, \quad \underbrace{z_{m+1}, \cdots, z_n}_{n-m=|V_{F_j}(nr')-U_{F_j}|} \quad \}$$

where $z_{m+1}, \cdots, z_n$ are random values generated by the simulator and they are uniformly distributed in the finite domain *Dom*. According to the theorems in [9], Net_1 cannot distinguish the distribution of Y_S's elements from that in

$$Y_R = \{\underbrace{(x_1)_{K_1}, \cdots, (x_m)_{K_1}}_{x_i \in V_{F_j}(nr') \cap U_{F_j}}, \underbrace{(x_{m+1})_{K_1}, \cdots, (x_n)_{K_1}}_{x_i \in V_{F_j}(nr') - U_{F_j}}\}$$

The Net_1's view of our protocol corresponds to Y_R. Therefore, the distribution of the Net_1's view of our protocol cannot be distinguished from this simulation. □

Next, we analyze the information learned by Net_1 and Net_2. After implementing our protocol, Net_1 knows the converted firewalls of FW_1 and FW_2, *e.g.*, Figure 4.4(j) and Figure 4.5(g), and Net_2 knows the comparison result, *e.g.*, Figure 4.6(d). On Net_1 side, for each field F_j ($1 \leq j \leq d$), it knows only $|U_{F_j}|$ and $V_{F_j}(nr') \cap U_{F_j}$, and it cannot reveal the rules of FW_2 for two reasons. First, in $V_{F_j}(nr') \cap U_{F_j}$, a numericalized prefix can be generated from many different numbers. For example, a prefix of IP addresses (32 bits) $b_1 b_2 \cdots b_k * \cdots *$ can be generated from 2^{32-k} different IP addresses. Second, even if Net_1 finds the number for a prefix in $V_{F_j}(nr') \cap U_{F_j}$, Net_1 doesn't know which rule in FW_2 contains that number. On Net_2 side, it only knows that the prefix x in $\mathcal{F}(x)$ belongs to which non-overlapping rules in FW_1. But such information is not enough to reveal the rules in FW_1.

4.5.2 Complexity Analysis

Let n_1 and n_2 be the number of rules in two adjacent firewalls FW_1 and FW_2, respectively, and d be the number of fields in both firewalls. For simplicity, we assume that the

numbers in different fields have the same length, say w bits. We first analyze the computation, space, and communication costs for the conversion of FW_1. Based on the theorem in [49], the maximum number of non-overlapping rules generated from the FDD is $(2n_1-1)^d$. Each non-overlapping rule consists of d w-bit intervals and each interval can be converted to at most $2w-2$ prefixes. Thus, the maximum number of prefixes generated from these non-overlapping rules is $d(2w-2)(2n_1-1)^d$. Note that the total number of prefixes cannot exceed 2^{w+1} because Net_1 puts all prefixes into one set. Thus, the computation cost of encryption by Net_1 is $\min(d(2w-2)(2n_1-1)^d, 2^{w+1})$. Therefore, for the conversion of FW_1, the computation cost of Net_1 is $\min(O(dwn_1^d), O(n_1^d + 2^w))$, the space cost of Net_1 is $O(dwn_1^d)$, the communication cost is $\min(O(dwn_1^d), O(2^w))$, and the computation cost of Net_2 is $\min(O(dwn_1^d), O(2^w))$. Similarly, for the conversion of FW_2, the computation cost of Net_2 is $\min(O(dwn_2^d), O(n_2^d + 2^w d))$, the space cost of Net_2 is $O(dwn_2^d)$, the communication cost is $\min(O(dwn_2^d), O(2^w d))$, and the computation cost of Net_1 is $\min(O(dwn_2^d), O(2^w d))$.

4.6 Experimental Results

We evaluate the effectiveness of our protocol on real firewalls and evaluate the efficiency of our protocol on both real and synthetic firewalls. We implemented our protocol using Java 1.6.0. Our experiments were carried out on a PC running Linux with 2 Intel Xeon cores and 16GB of memory.

4.6.1 Evaluation Setup

We conducted experiments over five groups of two real adjacent firewalls. Each firewall examines five fields, source IP, destination IP, source port, destination port, and protocol. The number of rules ranges from dozens to thousands. In implementing the commutative encryption, we used the Pohlig-Hellman algorithm [64] with a 1024-bit prime modulus and

160-bit encryption keys. To evaluate the effectiveness, we conducted our experiments over these five groups of adjacent firewalls. To evaluate the efficiency, for two firewalls in each group, we measured the processing time, the comparison time, and the communication cost.

Due to security concerns, it is difficult to obtain a large number of real adjacent firewalls. To further evaluate the efficiency, we generated a large number of synthetic firewalls based on Singh *et al.* 's method [74]. The synthetic firewalls also examine the same five fields as real firewalls. The number of rules in the synthetic firewalls ranges from 200 to 2000, and for each number, we generated 10 synthetic firewalls. To measure the efficiency, we first processed each synthetic firewall as FW_1 and then measured the processing time and communication cost of two parties. Second, we processed each synthetic firewall as FW_2 and measured the processing time and communication cost. Third, we measured the comparison time for every two synthetic firewalls. We did not evaluate the effectiveness of our protocol on synthetic firewalls because they are generated randomly and independently without considering whether two firewalls are adjacent or not.

4.6.2 Methodology

In this section, we define the metrics to measure the effectiveness of our protocol. Given our firewall optimization algorithm A, and two adjacent firewalls FW_1 and FW_2, we use $A(FW_1, FW_2)$ to denote a set of inter-firewall redundant rules in FW_2. Let $|FW|$ denote the number of rules in FW and $|A(FW_1, FW_2)|$ denote the number of inter-firewall redundant rules in FW_2. To evaluate the effectiveness, we define a *redundancy ratio* $\beta(A(FW_1, FW_2)) = \frac{|A(FW_1,FW_2)|}{|FW_2|}$. This ratio $\beta(A(FW_1, FW_2))$ measures what percentage of rules are inter-firewall redundant in FW_2.

4.6.3 Effectiveness and Efficiency on Real Policies

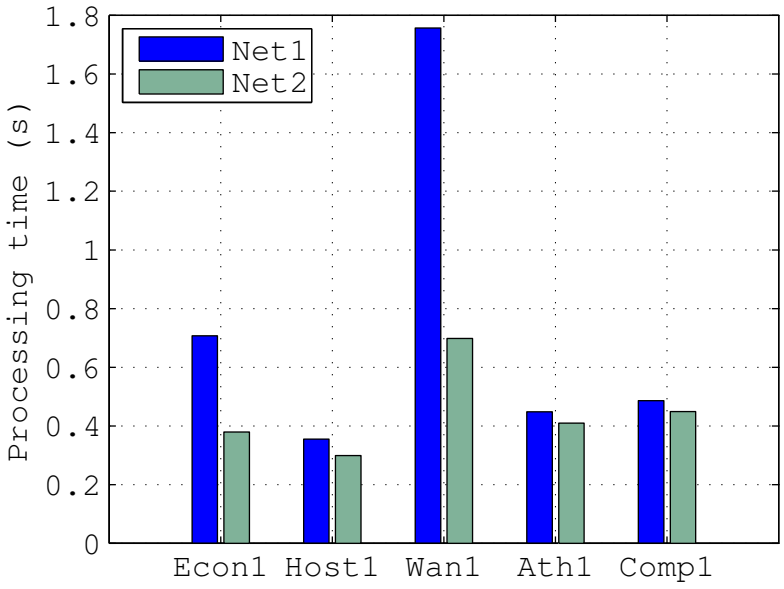
Table 4.1 shows the redundancy ratios for 5 real firewall groups. Column 1 shows the names of five real firewall groups. Columns 2 and 3 show the names of firewalls FW_1 and FW_2, respectively. Column 4 shows the number of rules in firewall FW_2. Figure 5.10 shows the processing time and communication cost of two parties Net_1 and Net_2 when processing FW_1; Figure 5.11 shows the processing time and communication cost of the two parties when processing FW_2. Figure 4.11 shows the comparison time of each group.

Group	FW_1	FW_2	$\|FW_2\|$	redundancy ratio
Econ	Econ1	Econ2	129	17.1%
Host	Host1	Host2	139	49.6%
Wan	Wan1	Wan2	511	1.0%
Ath	Ath1	Ath2	1308	14.4%
Comp	Comp1	Comp2	3928	14.7%

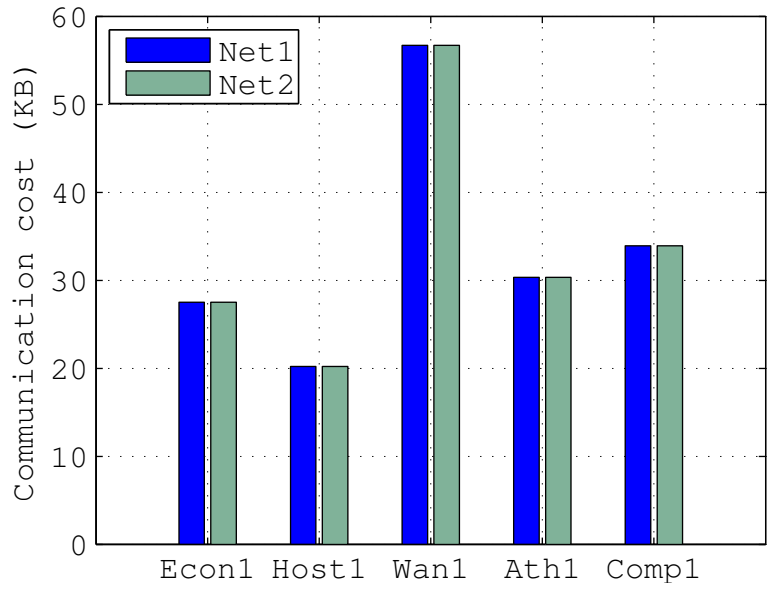
Table 4.1. Redundancy ratios for 5 real firewall groups

Our protocol achieves significant redundancy ratio on four real firewall groups. For 5 real firewall groups, our protocol achieves an average redundancy ratio of 19.4%. Particularly, for the firewall group Host, our protocol achieves 49.6% redundancy ratio, which implies that almost half of rules in Host2 are inter-firewall redundant rules. For firewall groups Econ, Ath, and Comp, our protocol achieves 14.4%-17.1% redundancy ratios, which implies that about 15% of rules in FW_2 are redundant in these three groups. Only for one firewall group Wan, our protocol achieves 1.0% redundancy ratio. We observed that most adjacent real firewalls have many inter-firewall redundant rules. Thus, our protocol can effectively remove inter-firewall redundant rules and significantly improve network performance.

Our protocol is efficient for processing and comparing two real firewalls. When processing FW_1 in the 5 real firewall groups, the processing time of Net_1 is less than 2 seconds and the processing time of Net_2 is less than 1 second. When processing FW_2 in those real firewall

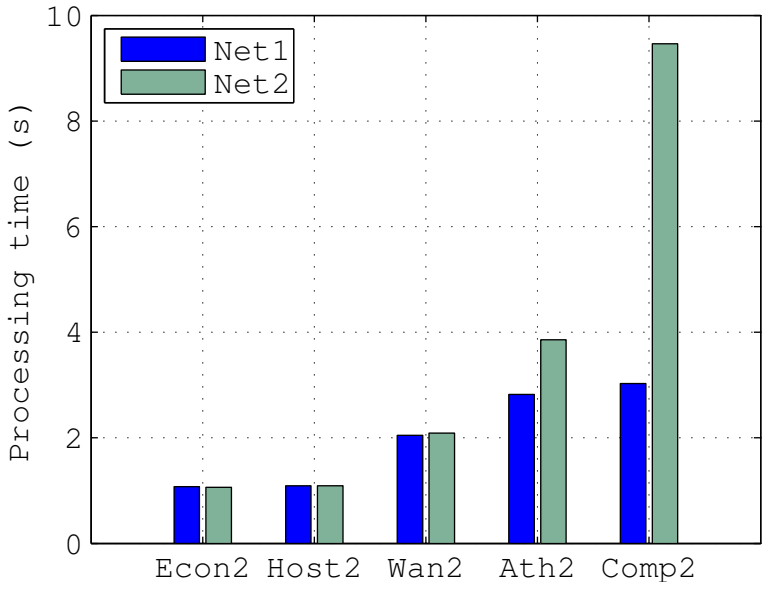


(a) Processing time

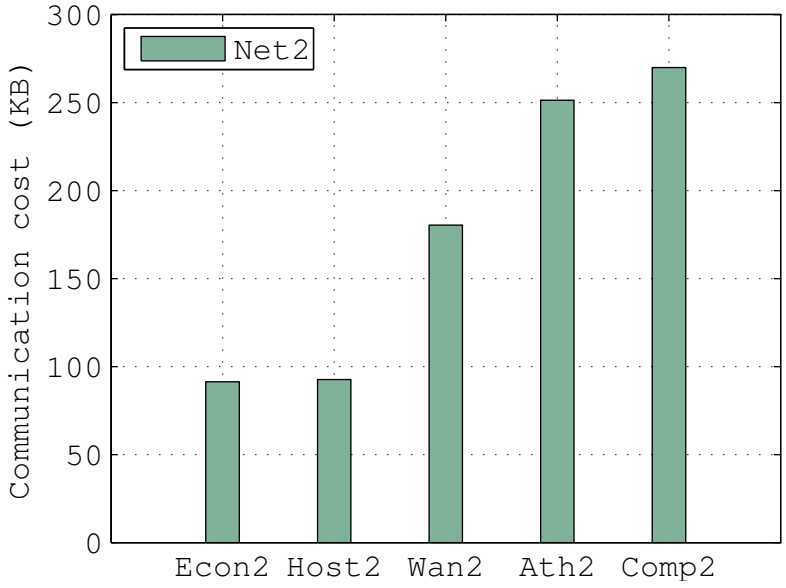


(b) Communication cost

Figure 4.9. Processing FW_1 on real firewalls



(a) Processing time



(b) Communication cost

Figure 4.10. Processing FW_2 on real firewalls

groups, the processing time of Net_1 is less than 4 seconds and the processing time of Net_2 is less than 10 seconds. The comparison time of two firewalls is less than 0.07 seconds. The total processing time of two parties is less than 15 seconds.

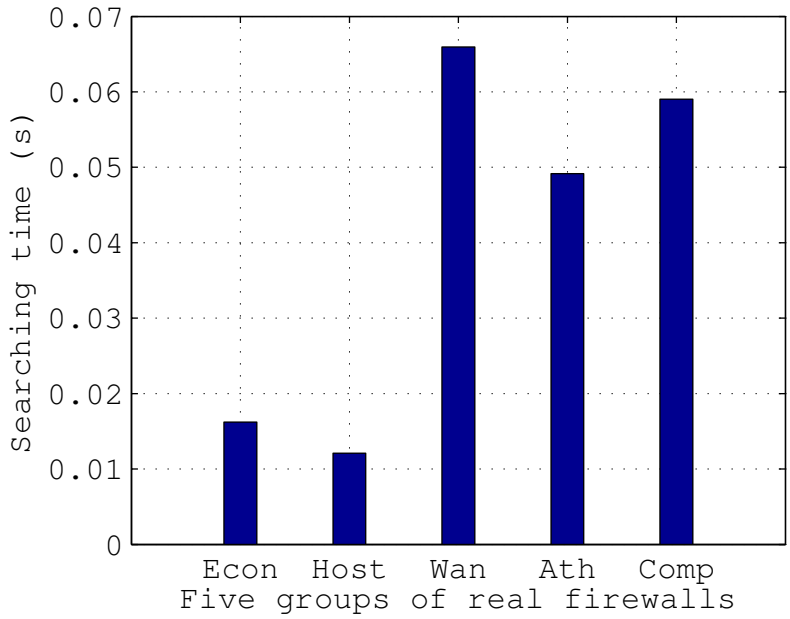
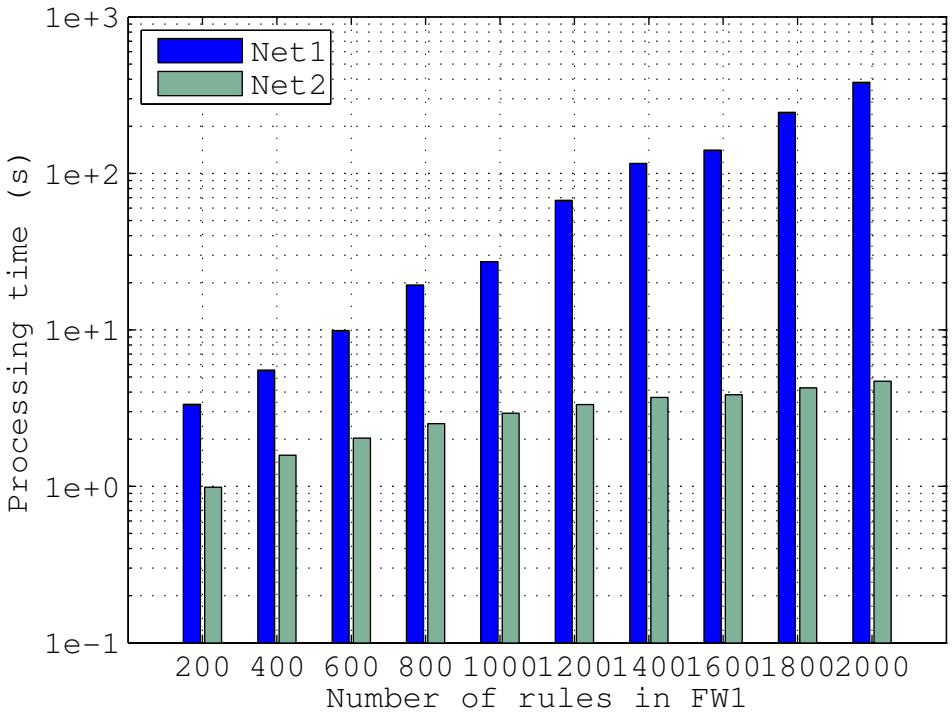
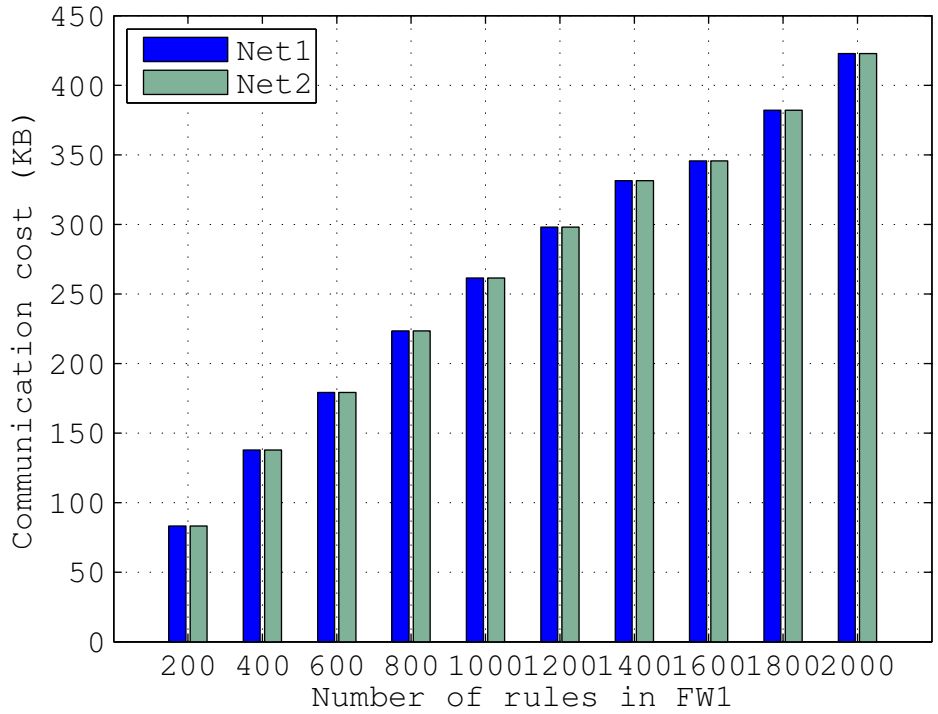


Figure 4.11. Comparing two real firewalls

Our protocol is efficient for the communication cost between two parties. When processing firewall FW_1 in the 5 real firewall groups, the communication cost from Net_1 to Net_2 and that from Net_2 to Net_1 are less than 60 KB. Note that the communication cost from Net_1 to Net_2 and that from Net_2 to Net_1 are the same because Net_1 and Net_2 encrypt the same number of values and the encrypted values have the same length, *i.e.*, 1024 bits in our experiments. When processing FW_2 in those real firewall groups, the communication cost from Net_2 to Net_1 is less than 300 KB. The total communication cost between two parties is less than 500 KB, which can be sent through the current network (*e.g.*, DSL network) around 10 seconds.

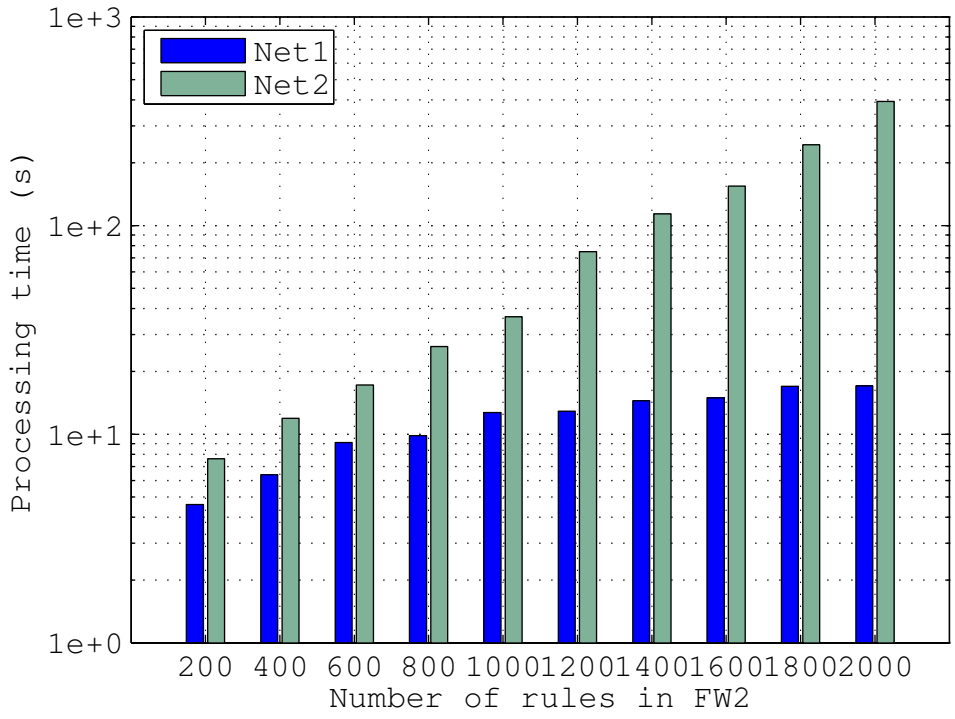


(a) Ave. processing time

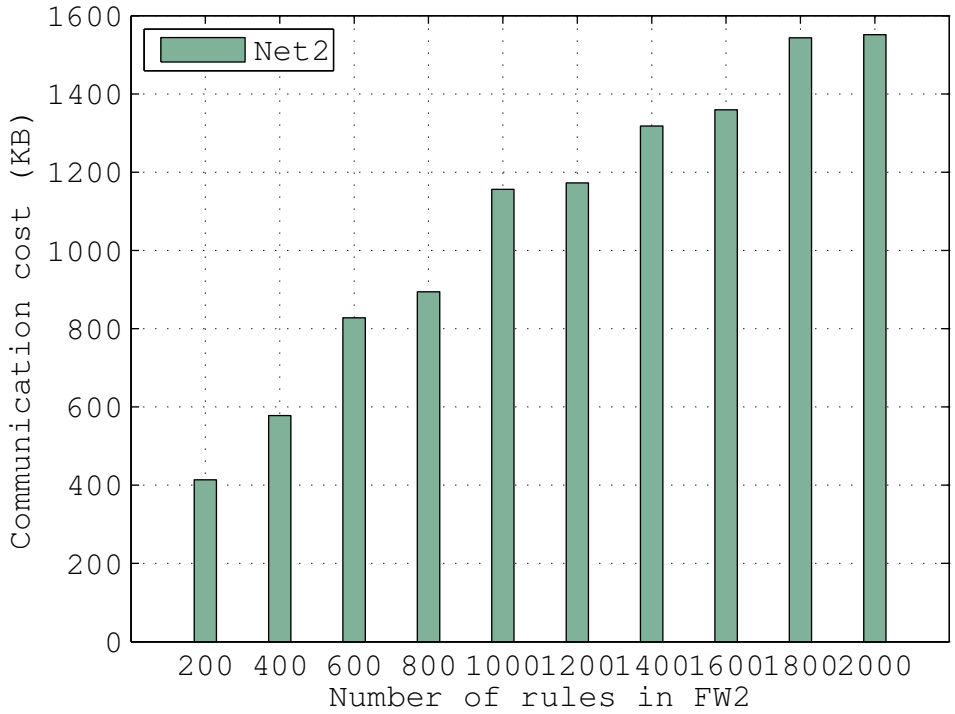


(b) Ave. communication cost

Figure 4.12. Processing FW_1 on synthetic firewalls



(a) Ave. processing time



(b) Ave. communication cost

Figure 4.13. Processing FW_2 on synthetic firewalls

4.6.4 Efficiency on Synthetic Policies

For the synthetic firewalls, Figure 5.12 and Figure 5.13 show the average processing time and communication cost of two parties Net_1 and Net_2 for processing FW_1 and FW_2, respectively. Figure 5.14 shows the average comparison time for every two synthetic firewalls. Note that the vertical axis of two figures 5.12(a) and 5.13(a) are in a logarithmic scale.

Our protocol is efficient for processing and comparing two synthetic firewalls. When processing the synthetic firewalls as FW_1, the processing time of Net_1 is less than 400 seconds and the processing time of Net_2 is less than 5 seconds. When processing the synthetic firewalls as FW_2, the processing time of Net_1 is less than 400 seconds and the processing time of Net_2 is less than 20 seconds. The comparison time of two synthetic firewalls is less than 4 seconds.

Our protocol is efficient for the communication cost between two synthetic firewalls. When processing the synthetic firewalls as FW_1, the communication cost from Net_1 to Net_2 and that from Net_2 to Net_1 grow linearly with the number of rules in FW_1, and both costs are less than 450 KB. Similarly, when processing synthetic firewalls as FW_2, the communication cost from Net_2 to Net_1 grows linearly with the number of rules in FW_2, and the communication cost from Net_2 to Net_1 is less than 1600 KB.

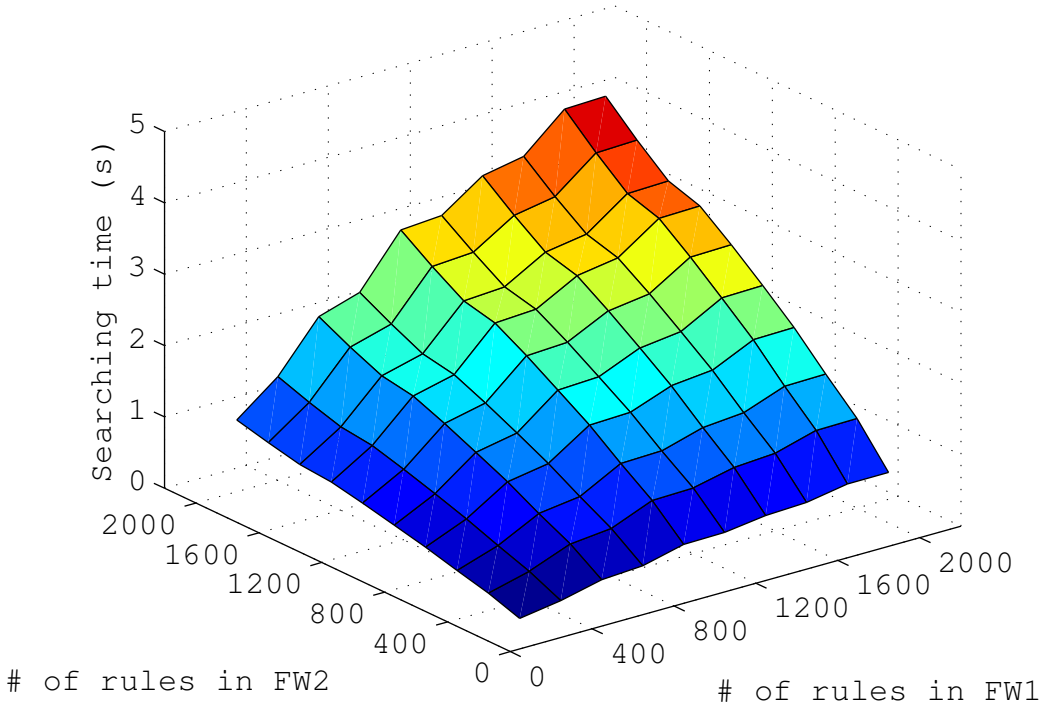


Figure 4.14. Comparing two synthetic firewalls

CHAPTER 5

Privacy Preserving Cross-Domain Network Reachability Quantification

5.1 Introduction

5.1.1 Background and Motivation

Network reachability quantification is important for understanding end-to-end network behavior and detecting the violation of security policies. Several critical concerns like router misconfiguration, policy violations and service availability can be verified through an accurate quantification. Network reachability for a given network path from the source subnet to the destination subnet is defined as the set of packets that are allowed by all network devices on the path. Quantifying network reachability is a difficult and challenging problem for two reasons. First, various complex mechanisms, such as Access Control Lists (ACLs), dynamic routing, and network address translation (NAT), have been deployed on network devices for restricting network reachability. Therefore, to perform an accurate analysis, administrators need to collect all the reachability restriction information from these network

devices. Collecting such information could be very difficult due to the privacy and security concerns. Second, the explosion of the Internet has caused an increase in the complexity and sophistication of these devices, thus, making reachability analysis computationally expensive and error-prone.

The current practice of reachability management is still "trial and error" due to the lack of network reachability analysis and quantification tools. Such practice leads to significant number of configuration errors, which has been shown to be the major cause of failure for Internet services [61]. Industry research also shows that a significant percentage of human effort and monetary resources are employed in maintaining the operational status of the network [43]. Several critical business applications and sensitive communication are affected severely due to network outages caused by misconfiguration errors. These events place a tremendous amount of pressure on network operators to debug the problems quickly. Thus, systematic analysis and quantification tools of network reachability are needed for understanding end-to-end network behavior and detecting configuration errors.

5.1.2 Limitation of Prior Art

Performing network reachability analysis is a complex task that involves aggregating and analyzing the reachability restriction information from all the devices along a given network path. The current practice of verifying reachability is to send probing packets. However, probing has two major drawbacks. First, probing is expensive to quantify network reachability because it needs to generate and send significant amount of packets. Second, probing is inaccurate, *e.g.*, it cannot probe the open ports with no server listening on them. Due to these drawbacks of probing, many approaches were proposed to address the reachability problem [12, 42, 44, 54, 75, 77]. The main assumption in all these approaches is that the reachability restriction information of each network device and other configuration state are

known to a central network analyst, who is quantifying the network reachability. However, in reality, it is common that the network devices along a given path belong to different parties where the reachability restriction information cannot be shared with others including the network analyst. Figure 5.1 shows a typical scenario of network reachability, where User_1 wants to know what packets he can send to User_2 through the given path. However, the network devices deployed along this path belong to three different parties, *i.e.*, S_1 and FW_1 belong to Subnet_1, FW_2, FW_3, and R_1 belong to ISP, FW_4 and S_2 belong to Subnet_2.

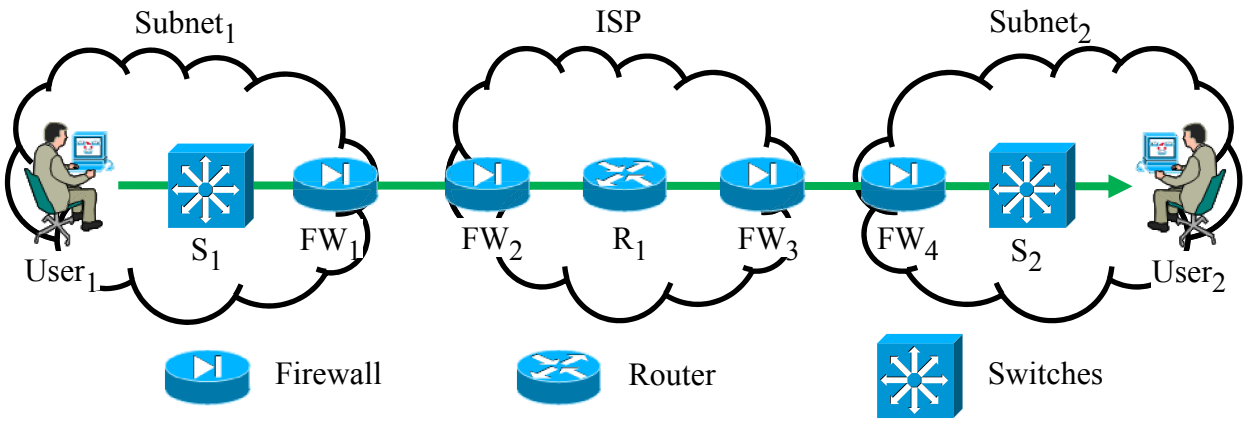


Figure 5.1. An example of end-to-end network reachability

Keeping the reachability restriction information private is important for two reasons. First, such information is often misconfigured and has security holes that can be exploited by attackers if it is disclosed. In reality, most firewall policies have security holes [76]. Disclosing ACLs allows attackers to analyze and utilize the vulnerabilities of subnets along a given path. For example, if ACLs along a path from Subnet_1 to Subnet_2 do not block some worm traffic, attackers can break into Subnet_2 from Subnet_1. In practice, neither ISPs nor private networks disclose their ACLs. Second, the reachability restriction information of a network device contains private information, *e.g.*, the ACLs of a network device contain the IP addresses of servers, which can be used by attacker to launch more targeted attacks. If such information of one device can be shared with other devices, an attacker needs to capture

only a single device (or a small subset) to know the security profile of the entire network, *i.e.*, the sensitive information in the ACLs can be abused for gaining profit or to disrupt important services across the network. In practice, even within an organization, often no employees other than the firewall administrators are allowed to access their firewall policies.

5.1.3 Cross-Domain Quantification of Network Reachability

To our best knowledge, no prior work has been proposed to address the problem of privacy-preserving network reachability quantification. We proposed the first privacy-preserving protocol for quantifying network reachability for a given network path across multiple parties. First, for the network devices belonging to each party, we convert the reachability restriction information of these devices to an access control list (ACL) by leveraging the existing network reachability quantification tool [44]. This tool takes as the input the reachability restriction information, including ACLs, all possible network transforms (*e.g.*, NAT and PAT), and protocol states (*e.g.*, connection-oriented and state-less), and outputs an ACL. Note that ACLs are the most important security component for network devices to filter the traffic. Considering the example in Figure 5.1, Figure 5.2 shows the three resulting ACLs, A_1, A_2, and A_3, for Subnet_1, ISP, and Subnet_2, respectively. For ease of presentation, in the rest of work, we use "ACL" to denote the resulting ACL converted from multiple ACLs as well as other reachability restriction information in one party. Second, we calculate the set of packets that are accepted by all the resulting ACLs on the given network path in a privacy-preserving manner. This calculation requires the comparison of the rules in all the ACLs, which is complex and error-prone.

Our proposed cross-domain quantification approach of network reachability can be very useful for many applications. Here we give two examples. First, a global view of the network reachability can help internet service providers (ISPs) to define better QoS policies. For

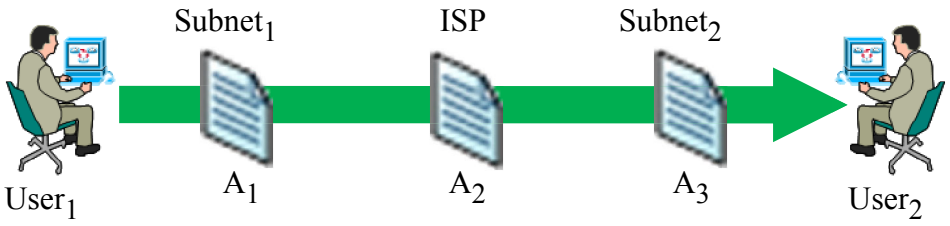


Figure 5.2. Three resulting ACLs converted from Figure 5.1

example, the knowledge of the different paths through which a particular type of traffic is allowed by the ACLs can help the ISPs to maintain a rated list of the best-quality paths in case of path failures. Second, since the network reachability is crucial for many internet companies, performing a privacy-preserving computation of the network reachability could become a new business for the ISPs and other parties that involve in this computation.

5.1.4 Technical Challenges

There are three key challenges in the privacy preserving quantification of network reachability. (1) It is computationally expensive. An ACL may consist of many rules, and each rule consists of multiple fields. Therefore, comparing multiple ACLs with a large number of rules can be quite expensive, even if only a few ACLs are involved in the process. Furthermore, the complexity of comparison can be expensive due to overlapping rules resulting in many comparisons. (2) Communication cost is high as even calculating the intersection of a small number of ACLs is a tedious process and requires a number of messages to be exchanged among different parties. (3) Protecting the privacy of the ACL rules is crucial. Since a rule has to be sent to other parties to enable comparison, it is necessary to propose a protocol that will not reveal the rule but still allows the different ACLs to calculate the intersection.

5.1.5 Our Approach

In this work, we propose the first cross-domain privacy-preserving protocol for quantifying network reachability. We consider n ACLs ($n \geq 2$) in a given network path and each ACL belongs to a distinct party. Starting with the destination ACL rules, our protocol calculates the intersection of these rules with the rules of the adjacent ACL. Next, using the results of this comparison, the adjacent ACL repeats the process with the next adjacent ACL until the source ACL is reached. At this point, the source ACL obtains the intersection of the rules from all ACLs along the given path. Briefly, our protocol consists of three phases: ACL preprocessing, ACL encoding and encryption, and ACL comparison.

In the first phase, we transform all the ACLs into an equivalent representation, Firewall Decision Diagram (FDD) [36], and then extract the non-overlapping rules with accept decisions. In the second phase, to perform privacy preserving comparison, we first transform the rules into a sequence of prefix numbers and then encrypt these numbers with secret keys of different parties. This phase enables different parties to compute the intersection of non-overlapping rules in their ACLs without revealing these rules. In the third phase, the destination ACL computes the intersection of its non-overlapping rules with the rules from its adjacent ACL, and then the adjacent ACL further repeats this computation with its adjacent ACL until the source ACL is reached. Note that the comparison result of every two adjacent ACLs is encrypted with multiple secret keys so that no party can reveal the comparison result independently. Finally, all the ACLs collaboratively decrypt the encrypted intersection of the non-overlapping rules, but only the first party (with the source ACL) obtains the result. This intersection represents the set of packets that are allowed by all ACLs on the given path.

5.1.6 Summary of Experimental Results

We performed extensive experiments over real and synthetic ACLs. Our experimental results show that the core operation of our protocol is efficient and suitable for real applications. The online processing time of an ACL with thousands of rules is less than 25 seconds and the comparison time of two ACLs is less than 5 seconds. The communication cost between two ACLs with thousands of rules is less than 2100 KB.

5.1.7 Key Contributions

We make three key contributions. (1) We propose the first cross-domain privacy-preserving protocol to quantify network reachability across multiple parties. Our protocol can accurately compute the intersection of the rules among the ACLs along a given network path without the need to share these rules across those parties. This is the first step towards privacy-preserving quantification of network reachability and it can be extended to other network metric measurements that are sensitive in nature. (2) We propose an optimization technique to reduce computation and communication costs. It reduces the number of ACL encryptions and the number of messages from $O(n^2)$ to $O(n)$. (3) We conducted extensive experiments on both real and synthetic ACLs and the result shows that our protocol is efficient and suitable for real applications.

5.2 Problem Statement and Threat Model

5.2.1 Problem Statement

We focus on quantifying the end-to-end network reachability for a given network path with multiple network devices belonging to different parties. The network devices are connected with physical interfaces for filtering outgoing packets and incoming packets. A network path

is a unidirectional path for transferring packets from the source to the destination. Along the given network path, there are multiple ACLs and other restriction information for filtering these packets. Multiple ACLs and other restriction information may belong to the same party. To convert them to a single ACL for one party, we first employ the existing network reachability approach, Quarnet [44], to convert them to reachability matrices. Second, we use the query language of Quarnet to obtain a set of packets that can pass through the given path within the party. Finally, we convert the set of packets to a single ACL. Without loss of generality, in the rest of work, we use the term "ACL" to denote the resulting ACL converted from multiple ACLs as well as other reachability restriction information in one party. Given an ACL A, let $M(A)$ denote the set packets that are accepted by A. Given a network path with n ACLs $A_1, A_2, ..., A_n$ for transferring packets from A_1 to A_n, where A_i belongs to the party P_i $(1 \leq i \leq n)$, quantifying the network reachability is computing the intersection among $M(A_1), \cdots, M(A_n)$, *i.e.*, $M(A_1) \cap M(A_2) \cdots \cap M(A_n)$.

In our context, we aim to design a privacy preserving protocol which enables the source ACL A_1 to compute the intersection of n ACLs $(n \geq 2)$, $M(A_1) \cap M(A_2) \cdots \cap M(A_n)$ without revealing rules in an ACL A_i $(1 \leq i \leq n)$ to any other party P_j $(j \neq i)$. We make the following four assumptions. (1) The destination of the network path cannot be an intermediate network device. In other words, the destination ACL A_n should filter the packets to end users but not to another network device. (2) The source ACL A_1 is not allowed to compute the intersection of a subset of all ACLs along the given network path. Because A_1 can easily reveal some rules in one ACL A_i by three steps. First, compute $M(A_1) \cap \cdots \cap M(A_{i-1})$. Second, compute $M(A_1) \cap \cdots \cap M(A_i)$. Third, compute $M(A_1) \cap \cdots \cap M(A_{i-1}) - M(A_1) \cap \cdots \cap M(A_i)$.

Note that if one party does not want to involve in this process or only wants to provide part of its ACL rules, the party P_1 can still run the protocol to compute network reachability among the remain ACLs. This requirement is very important especially for the party who really cares about the security of its private network, *e.g.*, a bank who will not share with

other parties the information that what packets can enter into its private network.

5.2.2 Threat Model

We consider the semi-honest model, where each party follows our protocol correctly but it may try to learn the ACL rules of other parties [35]. For example, the party P_1 may use the intermediate results to reveal the ACL rules of other parties. The semi-honest model is realistic in our context because a malicious party cannot gain benefits by providing a forged ACL or not following our protocol.

5.3 Privacy-Preserving Quantification of Network Reachability

To compute the network reachability from A_1 to A_n, our privacy-preserving protocol consists of three phases, *ACL preprocessing*, *ACL encoding and encryption*, and *ACL comparison*. In the first phase, ACL preprocessing, each party converts its ACL to a sequence of accepting rules. The union of the matching sets of these accepting rules is equal to the set of packets that are accepted by the ACL. In the second phase, ACL encoding and encryption, each party encodes and encrypts each field of its accepting rules for preserving the privacy of its ACL. In the third phase, ACL comparison, all parties compare their ACLs and finally the party P_1 finds out the set of packets that are accepted by all ACLs. Particularly, P_{n-1} compares the encoded and encrypted accepting rules from A_{n-1} with those from A_n, and finds out the multiple accepting rules whose union is equal to the intersection of $M(A_n)$ and $M(A_{n-1})$, $M(A_n) \cap M(A_{n-1})$. Then, P_{n-2} compares the accepting rules from ACL A_{n-2} with the resulting accepting rules in the first step, and finds out the multiple accepting rules whose union is equal to $M(A_n) \cap M(A_{n-1}) \cap M(A_{n-2})$. Repeat this step until P_1 finds out

the multiple accepting rules whose union is equal to $M(A_1) \cap \cdots \cap M(A_n)$. Note that, the resulting accepting rules of each step are in an encrypted format which prevents any party from revealing these rules by itself. To reveal the final accepting rules, P_1 requires all other parties to decrypt these rules with their private keys and then P_1 decrypts these rules.

The basic problem of privacy-preserving network reachability is how to compute the intersection among multiple range rules belonging to different parties in a privacy preserving manner. This problem boils down to the problem of computing intersection of two ranges $[a, b]$ and $[a', b']$, denoted as $[a, b] \cap [a', b']$. Thus, we first describe the privacy-preserving protocol for computing $[a, b] \cap [a', b']$, and then describe the three phases in our network reachability protocol.

5.3.1 Privacy-Preserving Range Intersection

To compute the intersection of a range $[a, b]$ from A_i and a range $[a', b']$ from A_j, our basic idea is to check which range among $[min, a-1]$, $[a, b]$, and $[b+1, max]$ includes a' or b', where min and max are the minimum and maximum numbers, respectively. Thus, the problem of computing $[a, b] \cap [a', b']$ boils down to the problem of checking whether a number is in a range, *e.g.*, $a' \in [min, a-1]$, which can be solved by leveraging the prefix membership verification scheme in [48]. The idea of prefix membership verification is to convert the problem of checking whether a number is in a range to the problem of checking whether two sets have common elements. Our scheme consists of six steps:

(1) The party P_i converts range $[a, b]$ to three ranges $[min, a-1]$, $[a, b]$, and $[b+1, max]$, where min and max are the minimum and maximum numbers of the corresponding field's domain, respectively. For example, [5,7] is converted to [0,4], [5,7], and [8,15], where 0 and 15 are the minimum and maximum numbers. Note that $[min, a-1]$ and $[b+1, max]$ may not exist. If $a=min$, then $[min, a-1]$ does not exist; if $b=max$, then $[b+1, max]$ does not exist.

(2) The party P_i converts each range to a set of prefixes, whose union corresponds to the range. Let $\mathcal{S}([min, a-1])$, $\mathcal{S}([a,b])$, and $\mathcal{S}([b+1, max)$ denote the resulting prefixes for the three ranges, respectively. For example, $\mathcal{S}([5, 7])$={0101, 011*}, where "*" denotes that this bit can be 0 or 1.

(3) The party P_j generates the *prefix families* of a and b, denoted as $\mathcal{F}(a)$ and $\mathcal{F}(b)$. The *prefix family* $\mathcal{F}(a)$ consists of a and all the prefixes that contains a. Assuming w is the bit length of a, $\mathcal{F}(a)$ consists of $w+1$ prefixes where the l-th prefix is obtained by replacing the last $l-1$ bits of a by $*$. For example, as the binary representation of 6 is 0110, we have $\mathcal{F}(6)$={0110, 011*, 01**, 0***, ****}. It is easy to prove that $a' \in [a,b]$ if and only if $\mathcal{F}(a') \cap \mathcal{S}([a,b]) \neq \emptyset$.

(4) Two parties P_i and P_j convert the resulting prefixes to numbers so that they can encrypt them in the next step. We use the prefix numericalization scheme in [20]. This scheme basically inserts 1 before $*$s in a prefix and then replaces every $*$ by 0. For example, 01** is converted to 01100. If the prefix does not contain $*$s, we place 1 at the end of the prefix. For example, 1100 is converted to 11001. Given a set of prefixes S, we use $\mathcal{N}(S)$ to denote the resulting set of numericalized prefixes. Thus, $a' \in [a,b]$ if and only if $\mathcal{N}(\mathcal{F}(a')) \cap \mathcal{N}(\mathcal{S}([a,b])) \neq \emptyset$.

(5) Checking whether $\mathcal{N}(\mathcal{F}(a')) \cap \mathcal{N}(\mathcal{S}([a,b])) \neq \emptyset$ is basically checking whether an element from $\mathcal{N}(\mathcal{F}(a'))$ is equal to an element from $\mathcal{N}(\mathcal{S}([a,b]))$. We use commutative encryption (*e.g.*, [64, 67]) to do this checking. Given a number x and two encryption keys K_i and K_j, a commutative encryption satisfies the property $((x)_{K_i})_{K_j} = ((x)_{K_j})_{K_i}$, *i.e.*, encryption with K_i and then K_j is equivalent to encryption with K_j and then K_i. For ease of presentation, we use $(x)_{K_{ij}}$ to denote $((x)_{K_i})_{K_j}$. In our scheme, to check whether $\mathcal{N}(\mathcal{F}(a')) \cap \mathcal{N}(\mathcal{S}([a,b])) \neq \emptyset$, P_i first encrypts numbers in $\mathcal{N}(\mathcal{S}([a,b]))$ with its private key K_i, then P_j further encrypts them by its private key K_j and sends them back to P_i. Let $\mathcal{N}(\mathcal{S}([a,b]))_{K_{ij}}$ denote the result. Second, P_j encrypts numbers in $\mathcal{N}(\mathcal{F}(a'))$ with K_j and then P_i encrypts them by K_i. Let

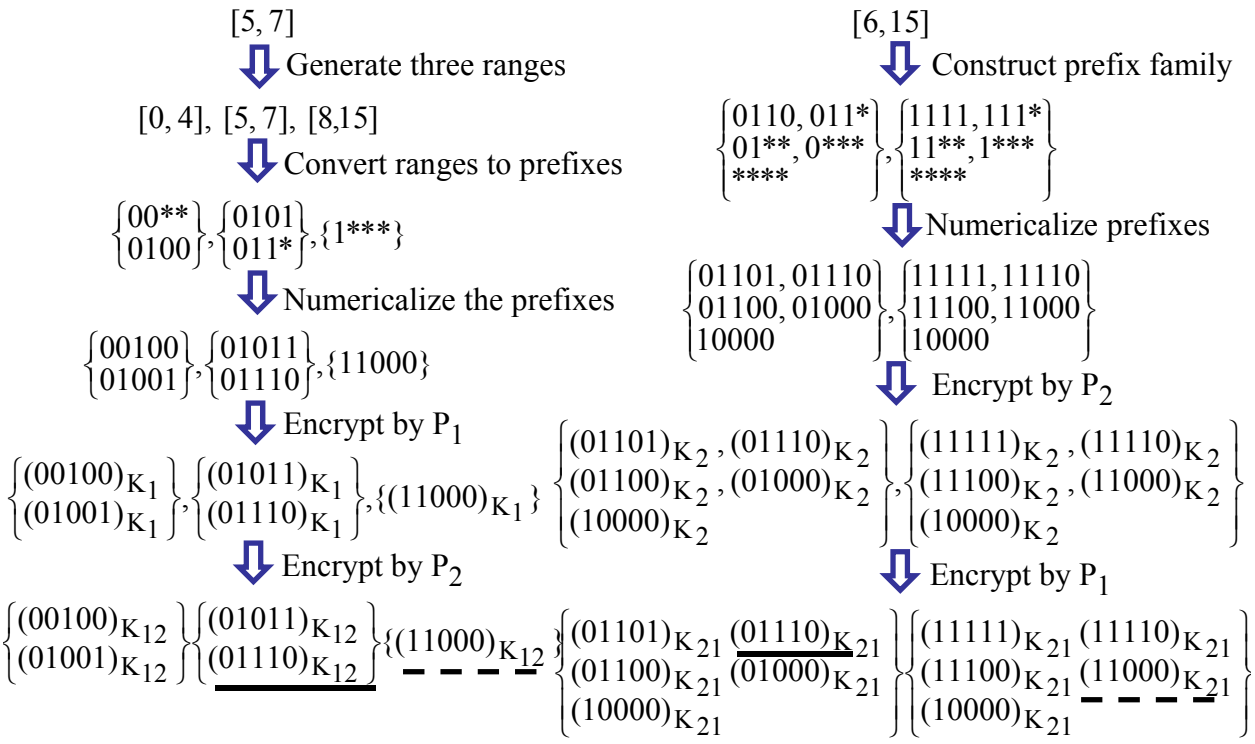


Figure 5.3. Privacy-preserving range intersection

$\mathcal{F}(a')_{K_{ji}}$ denote the result. Finally, P_i can check whether there is a common element in two sets $\mathcal{N}(\mathcal{S}([a,b]))_{K_{ij}}$ and $\mathcal{F}(a')_{K_{ji}}$.

Through the previous steps, P_i knows that which range among $[min, a-1]$, $[a,b]$, and $[b+1, max]$ includes a' or b'. Based on this information, P_i can compute $[a,b] \cap [a',b']$. For example, if $a' \in [min, a-1]$ and $b' \in [a,b]$, $[a,b] \cap [a',b'] = [a,b']$. Note that a' and b' are in the form of $\mathcal{F}(a')_{K_{ji}}$ and $\mathcal{F}(b')_{K_{ji}}$. P_i cannot reveal a' and b' without knowing P_j's private key K_j. Figure 5.3 illustrates the process of computing the intersection of $[5,7]$ (from A_1) and $[6,15]$ (from A_2).

5.3.2 ACL Preprocessing

In the ACL preprocessing phase, each party P_i ($1 \leq i \leq n$) computes the set of packets $M(A_i)$ that are accepted by its ACL A_i. P_i first converts its ACL to an equivalent sequence of

non-overlapping rules. Non-overlapping rules have an important property, that is, for any two non-overlapping rules nr and nr', the intersection of the two corresponding matching sets is empty, *i.e.*, $M(nr) \cap M(nr') = \emptyset$. Thus, any packet p matches one and only one non-overlapping rule converted from A_i and the decision of this non-overlapping rule is the decision of A_i for the packet p. Therefore, instead of computing the set $M(A_i)$, P_i only needs to retrieve all the non-overlapping accepting rules because the union of the matching sets of these rules is equal to $M(A_i)$. The preprocessing of each A_i includes three steps:

(1) P_i converts its ACL A_i to an equivalent acyclic directed graph, called firewall decision diagram (FDD) [36]. An FDD construction algorithm is presented in [49]. Figure 5.4(b) shows the FDD constructed from Figure 5.4(a).

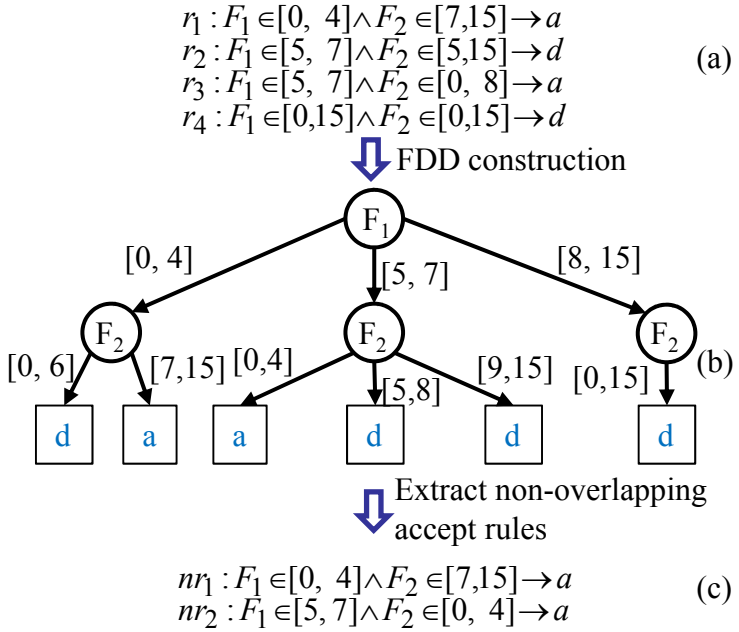


Figure 5.4. The Conversion of A_1

(2) P_i extracts non-overlapping accepting rules from the FDD. We do not consider non-overlapping discarding rules because the packets discarded by any ACL A_i cannot pass through the path. Figure 5.4(c) shows the non-overlapping accepting rules extracted from the FDD in Figure 5.4(b).

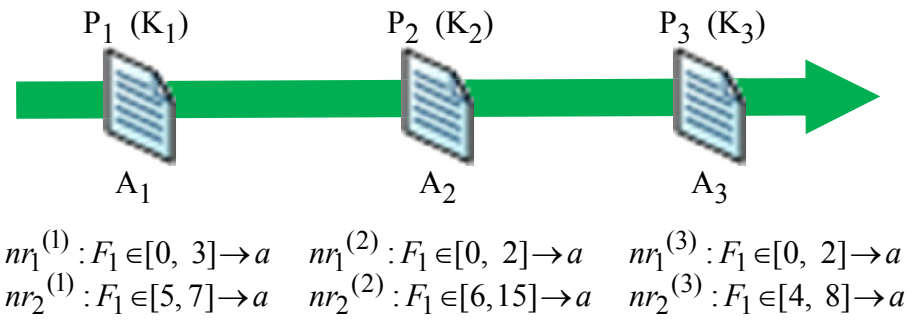


Figure 5.5. The example three adjacent ACLs

After ACL preprocessing, the problem of privacy preserving quantification of network reachability boils down to the problem of privacy preserving range intersection, which is in fact the basic problem in this work. Next, P_1 needs to compare its accepting rules from A_1 with those from other $n-1$ ACLs. Without loss of generality, in the next two subsections, we use a simplified example in Figure 5.5 to show that how to compute the network reachability among three ACLs. Each ACL has only one field and the domain for the field is [0,15]. Note that in Figure 5.5, $nr_1^{(i)}$ and $nr_2^{(i)}$ denote two non-overlapping accepting rules for ACL A_i ($1 \leq i \leq 3$). Obviously, the network reachability among these three ACLs can be denoted as two accepting rules $F_1 \in [0, 2] \rightarrow a$ and $F_1 \in [6, 7] \rightarrow a$. Next, we will show that how to compute these two rules in a privacy preserving manner.

5.3.3 ACL Encoding and Encryption

In the ACL encoding and encryption phase, all parties need to convert their non-overlapping accepting rules to another format such that they can collaboratively compute the network reachability while one party cannot reveal the ACL of any other party. To achieve this purpose, each party first encodes its non-overlapping accepting rules and then encrypts each field of these rules. Recall the privacy-preserving range intersection scheme in Section 5.3.1, two parties employ different encoding methods, one converts a range $[a, b]$ to a set of prefixes $\mathcal{S}([a, b])$, and another converts a number a' to its prefix family $\mathcal{F}(a')$. Thus, in this

phase, there are two different encoding and encryption methods for different ACLs. Assume that each party P_i ($1 \leq i \leq n$) has a private key K_i. Each party P_j ($1 \leq j \leq n-1$) encodes the non-overlapping accepting rules from A_j by converting each range to a set of prefixes and then encrypts each numericalized prefix by other parties P_j, P_{j+1}, $\cdots$, P_n. Let $\mathcal{H}$ denote the encoding function used by the party P_j ($1 \leq j \leq n-1$). Let $F_1 \in [a_1, b_1] \wedge \cdots \wedge F_d \in [a_d, b_d]$ denote the predicate of an accepting rule over d fields. The encoding and encryption result of this accepting rule for A_j is $\mathcal{H}_{K_{j \cdots n}}([a_1, b_1]) \wedge \cdots \wedge \mathcal{H}_{K_{j \cdots n}}([a_d, b_d])$. The party P_n encodes the non-overlapping accepting rules from A_n by converting each range to two prefix families and then encrypts each numericalized prefix by itself. Let $\mathcal{L}$ denote the encoding function used by the party P_n. Considering the above accepting rule, the result is $\mathcal{L}_{K_n}([a_1, b_1]) \wedge \cdots \wedge \mathcal{L}_{K_n}([a_d, b_d])$. We discuss the procedure of these two encoding and encryption methods in detail as follows.

Encoding and Encryption of ACL A_j ($1 \leq j \leq n-1$)

(1) For each non-overlapping accepting rule $F_1 \in [a_1, b_1] \wedge \cdots \wedge F_d \in [a_d, b_d]$, P_j converts each range $[a_l, b_l]$ ($1 \leq l \leq d$) to three ranges $[min_l, a_l - 1]$, $[a_l, b_l]$, $[b_l + 1, max_l]$, where min_l and max_l are the minimum and maximum values for the l-th field, respectively. Figure 5.6(b) shows the ranges generated from Figure 5.6(a).

(2) P_j converts each range to a set of prefixes. Figure 5.6(c) shows the prefixes generated from Figure 5.6(b). That is, for the three ranges converted from $[a_l, b_l]$, compute $\mathcal{S}([min_l, a_l - 1])$, $\mathcal{S}([a_l, b_l])$, $\mathcal{S}([b_l + 1, max_l])$.

(3) P_j unions all these prefix sets and permutes these prefixes. Figure 5.6(d) shows the resulting prefix set. This step has two benefits. First, it avoids encrypting and sending duplicate prefixes, and hence, significantly reduces the computation and communication costs for the next two steps. Second, it enhances the security, any other parties except P_j cannot reconstruct the non-overlapping accepting rules, because it is difficult to correlate the prefixes to their corresponding rules without the knowledge of the original ACL.

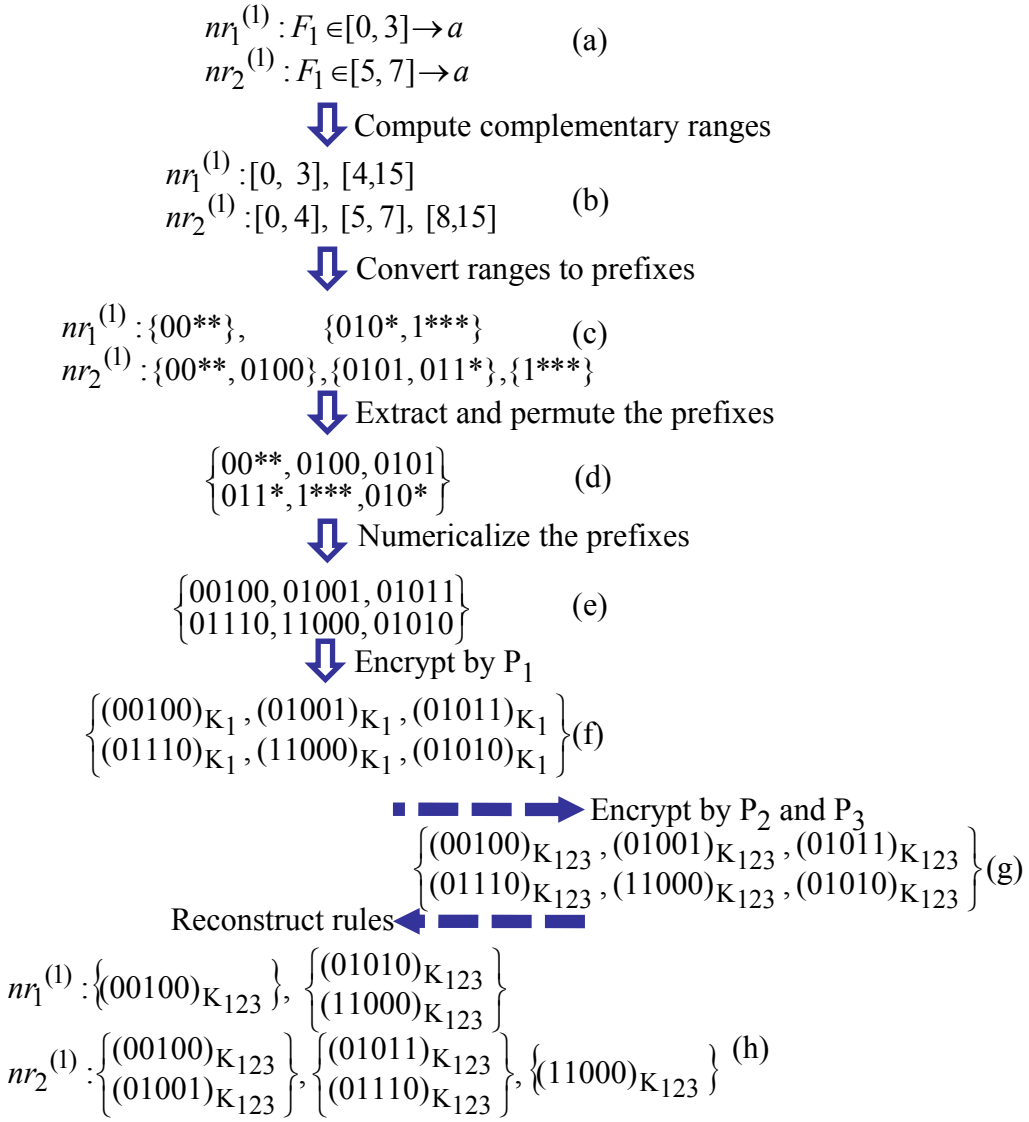


Figure 5.6. Encoding and encryption of ACL A_1

(4) P_j numericalizes and encrypts each prefix using K_j. Figure 5.6(e) and 5.6(f) show the numericalized and encrypted prefixes.

(5) P_j sends the resulting prefixes to P_{j+1} which further encrypts them with its private key K_{j+1}. Then, P_{j+1} sends the result prefixes to P_{j+2} which further encrypts them with K_{j+2}. This process is repeated until P_n encrypts them. Finally, P_n sends to P_j the resulting prefixes that are encrypted $n-j+1$ times. Figure 5.6(f) shows the result after encrypting by P_1 and Figure 5.6(g) shows the result after encrypting by P_2 and P_3.

(6) P_j reconstructs the non-overlapping accepting rules from the multiple encrypted prefixes because P_j knows which prefix belongs to which field of which rule.

Based on the above steps, the encoding and encryption function used by P_j $(1 \leq j \leq n-1)$ is defined as $\mathcal{H}_{K_{j\cdots n}}([a_l, b_l]) = (\mathcal{N}(\mathcal{S}(min_l, a_l - 1))_{K_{j\cdots n}}, \mathcal{N}(\mathcal{S}(a_l, b_l))_{K_{j\cdots n}}, \mathcal{N}(\mathcal{S}(b_l + 1, max_l))_{K_{j\cdots n}})$, where $[a_l, b_l]$ is the range in the l-th field of a rule in A_j. Figure 5.8(a) illustrates the encoding and encryption result of ACL A_2 in Figure 5.5. The only difference between operations for A_1 and A_2 is that A_1's numericalized prefixes are encrypted by all the three parties while A_2's numericalized prefixes are only encrypted by P_2 and P_3.

Encoding and Encryption of ACL A_n

(1) For each range $[a_l, b_l]$ of a non-overlapping rule, P_n generates two prefix families $\mathcal{F}(a_l)$ and $\mathcal{F}(b_l)$. Figure 5.7(b) shows the result from Figure 5.7(a).

(2) P_n numericalizes and encrypts the prefixes using its private key K_n. Figure 5.7(c) shows the resulting prefixes.

Based on these steps, the encoding and encryption function used by A_n is defined as

$$\mathcal{L}_{K_n}([a_l, b_l]) = (\mathcal{N}(\mathcal{F}(a_l))_{K_n}, \mathcal{N}(\mathcal{F}(b_l))_{K_n})$$

where $[a_l, b_l]$ is the range in the l-th field of a rule.

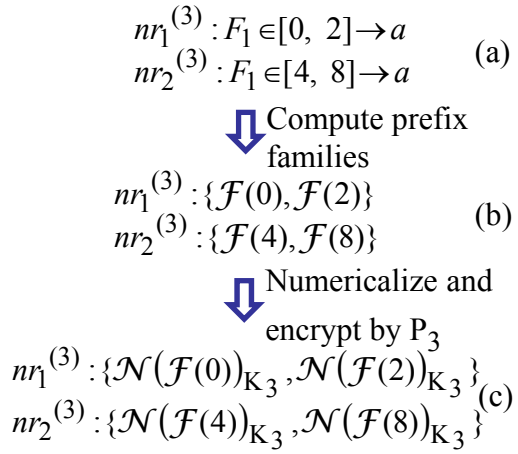


Figure 5.7. Encoding and encryption of ACL A_3

5.3.4 ACL Comparison

After the first two phases, each party P_j $(1 \leq j \leq n-1)$ converts its ACL A_j to a sequence of encrypted non-overlapping accepting rules and P_n converts its ACL A_n to a sequence of encrypted numbers. Figure 5.8(a) shows the encrypted non-overlapping rules converted from A_2 in Figure 5.5 and Figure 5.8(b) shows the encrypted numbers converted from A_3 in Figure 5.5.

In the ACL comparison phase, we need to compare the n sequences of encrypted non-overlapping accepting rules or encrypted numbers from every two adjacent ACLs. Without loss of generality, we only present the comparison between A_{n-1} and A_n. This comparison includes four steps:

(1) P_n sends the resulting sequence to P_{n-1} which further encrypts them with its private key K_{n-1}. Let $\mathcal{L}_{K_n}([a'_1, b'_1]) \wedge \cdots \wedge \mathcal{L}_{K_n}([a'_d, b'_d])$ denote the encoded and encrypted result of the accepting rule nr' from A_n. P_{n-1} encrypts it with K_{n-1}, *i.e.*, computes $\mathcal{L}_{K_{n(n-1)}}([a'_1, b'_1]) \wedge \cdots \wedge \mathcal{L}_{K_{n(n-1)}}([a'_d, b'_d])$. Figure 5.8(c) shows the encrypted result from Figure 5.8(b).

(2) For each non-overlapping accepting rule nr from A_{n-1}, P_{n-1} computes $nr \cap nr'$. Let $\mathcal{H}_{K_{(n-1)n}}([a_1, b_1]) \wedge \cdots \wedge \mathcal{H}_{K_{(n-1)n}}([a_d, b_d])$ denote the encoded and encrypted result of

nr. To compute $nr \cap nr'$, for each field l ($1 \leq l \leq d$), P_{n-1} compares $\mathcal{H}_{K_{(n-1)n}}([a_l, b_l])$ with $\mathcal{L}_{K_n(n-1)}([a'_l, b'_l])$ where

$\mathcal{H}_{K_{(n-1)n}}([a_l, b_l]) = (\mathcal{N}(\mathcal{S}(min_l, a_l - 1))_{K_{(n-1)n}}$,
$\mathcal{N}(\mathcal{S}(a_l, b_l))_{K_{(n-1)n}}, \mathcal{N}(\mathcal{S}(b_l + 1, max_l))_{K_{(n-1)n}})$

$\mathcal{L}_{K_{n(n-1)}}([a'_l, b'_l]) = (\mathcal{N}(\mathcal{F}(a'_l))_{K_{n(n-1)}}, \mathcal{N}(\mathcal{F}(b'_l))_{K_{n(n-1)}})$. According to the privacy-preserving range intersection, to check whether $a'_l \in [min_l, a_l - 1]$, P_{n-1} checks whether $\mathcal{N}(\mathcal{S}(min_l, a_l - 1))_{K_{(n-1)n}} \cap \mathcal{N}(\mathcal{F}(a'_l))_{K_{n(n-1)}} = \emptyset$. Similarly, P_{n-1} checks whether $a'_l \in [a_l, b_l]$, $a'_l \in [b_l + 1, max_l]$, and whether $b'_l \in [min_l, a_l - 1]$, $b'_l \in [a_l, b_l]$, $b'_l \in [b_l + 1, max_l]$. Based on the above result, P_{n-1} computes the intersection between $[a_l, b_l]$ and $[a'_l, b'_l]$, *i.e.*, $[a_l, b_l] \cap [a'_l, b'_l]$. Let T_l denote $[a_l, b_l] \cap [a'_l, b'_l]$. For example, if $a'_l \in [a_l, b_l]$ and $b'_l \in [b_l + 1, max_l]$, the condition $a_l \leq a'_l \leq b_l < b'_l$ holds and hence $T_l = [a'_l, b_l]$. If for any T_l ($1 \leq l \leq d$) the condition $T_l \neq \emptyset$ holds, then $nr \cap nr' = T_1 \wedge \cdots \wedge T_d$.

Note that the party P_{n-1} cannot reveal a'_l and b'_l through this comparison because P_{n-1} doesn't know P_n's private key K_n. Thus, if $T_l = [a'_l, b_l]$, P_{n-1} only knows $\mathcal{N}(\mathcal{F}(a'_l))_{K_{n(n-1)}}$ and b_l. We denote the information that P_{n-1} knows about T_l as $\{\mathcal{N}(\mathcal{F}(a'_l))_{K_{n(n-1)}}, b_l\}$. Figure 5.8(d) shows the result after comparing A_2 and A_3. Note that the result may contain the numbers from A_{n-1}'s non-overlapping accepting rules, which are not encrypted, *e.g.*, the number 6 in Figure 5.8(d).

(3) To preserve the privacy of A_{n-1}, the party P_{n-1} encodes and encrypts the numbers from A_{n-1}'s non-overlapping accepting rules, and then sends the result to P_n. For example, for a number a_l, P_{n-1} computes $\mathcal{N}(\mathcal{F}(a_l))_{K_{(n-1)}}$. Figure 5.8(e) shows the result after encoding and encrypting 6.

(4) To facilitate the next comparison with A_{n-2}, P_{n-1} sends the comparison result to P_n and then P_n encrypts the numbers from A_{n-1}'s non-overlapping accepting rules. Figure 5.8(f) shows the encryption result.

Repeat these four steps to further compare A_{n-2} with the result stored in P_n. After

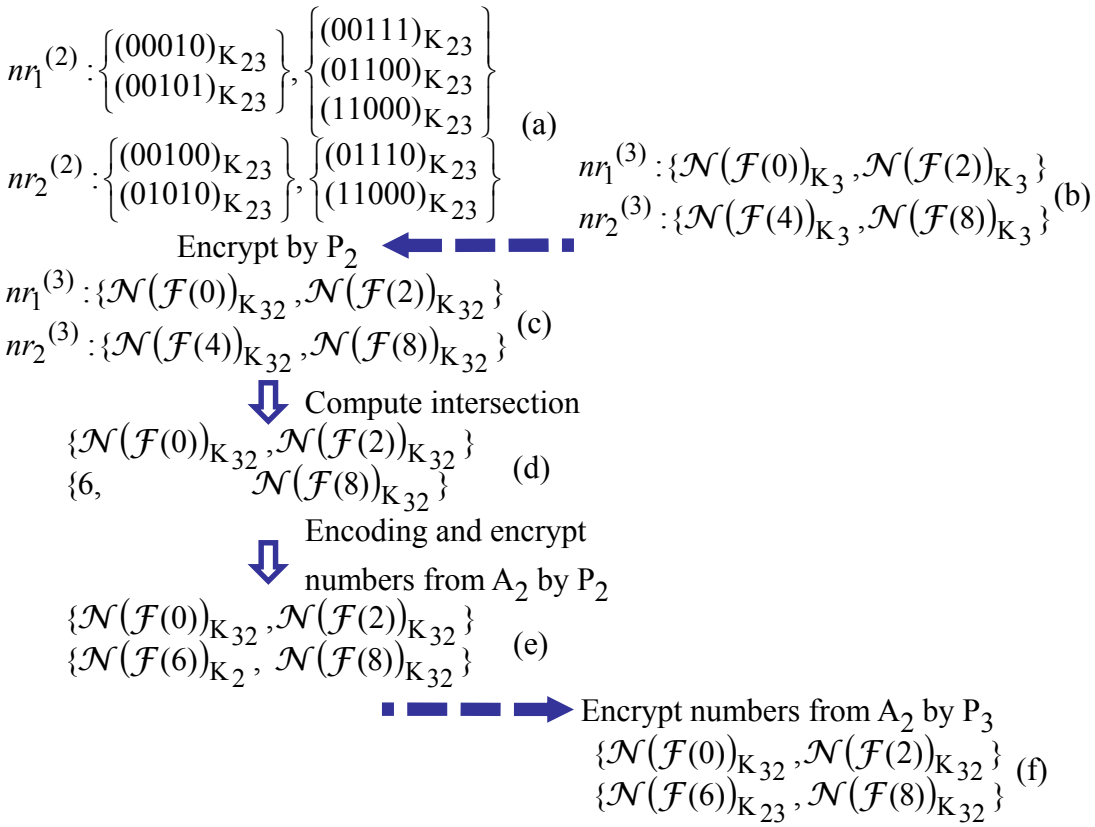


Figure 5.8. Comparison of ACLs A_2 and A_3

all parties finish the comparison, P_n has the comparison result. Let $\{\mathcal{N}(\mathcal{F}(a_l))_{K_{1\cdots n}}$, $\mathcal{N}(\mathcal{F}(b_l))_{K_{1\cdots n}}\}$ denote the l-th field of an encrypted rule in the result. To decrypt this rule, P_n first decrypts it with K_n and sends to P_{n-1}. Then, P_{n-1} decrypts it with K_{n-1} and sends to P_{n-2}. Repeat this step until P_1 decrypts the rule. Finally, P_1 have the result $F_1 \in [a_1, b_1] \wedge \cdots \wedge F_d \in [a_d, b_d]$. The comparison result of three ACLs in Figure 5.5 is $\{\mathcal{N}(\mathcal{F}(0))_{K_{123}}, \mathcal{N}(\mathcal{F}(2))_{K_{123}}\}$, and $\{\mathcal{N}(\mathcal{F}(6))_{K_{123}}, \mathcal{N}(\mathcal{F}(7))_{K_{123}}\}$. Figure 5.9 shows the decryption process of the comparison result.

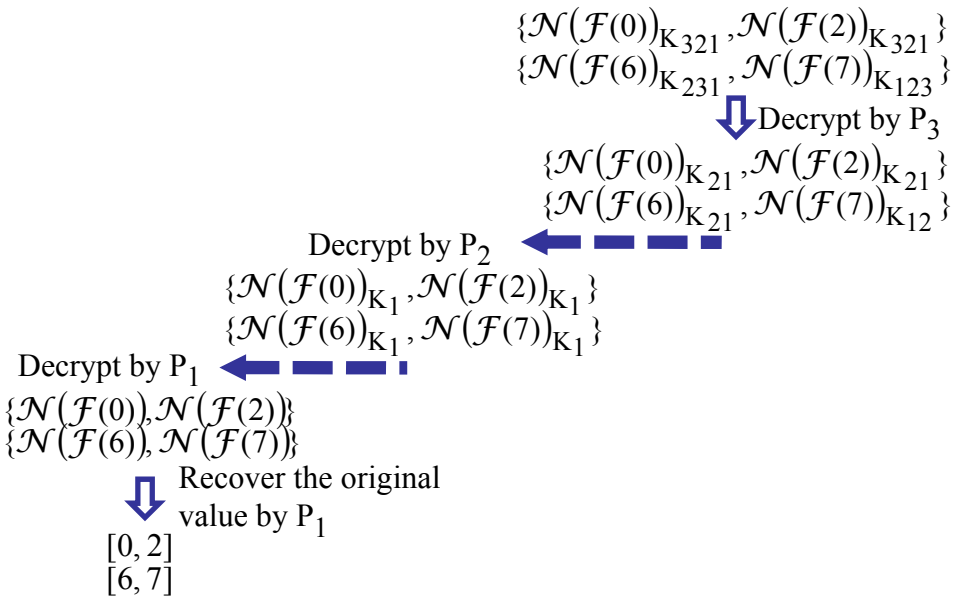


Figure 5.9. Decryption process of the comparison result

5.4 Security and Complexity Analysis

5.4.1 Security Analysis

The security of our protocol is based on the two important properties of the commutative encryption. (1) *Secrecy*: for any x and key K, given $(x)_K$, it is computationally infeasible to compute K. (2) *Indistinguishability*: the distribution of $(x)_K$ is indistinguishable from the distribution of x. Based on the first property, without knowing P_j's secret key K_j, the party P_i $(i \neq j)$ cannot decrypt the encrypted numbers from P_i. Furthermore, one party P_i cannot statistically analyze encrypted numbers from A_j $(i \neq j)$ because each party P_j $(1 \leq j \leq n-1)$ unions the encrypted prefix numbers into one set before sending them to P_i for further encryption. Therefore, after the first and second phases of our protocol (*i.e.*, *ACL preprocessing* and *ACL encoding and encryption*), the party P_i cannot reveal the ACL of any other party. Based on the second property, we can prove that after the third phase (*i.e.*, *ACL comparison*), the party P_i only learns the limited information of the ACLs of other parties,

but such information cannot help it reveal them. Without loss of generality, we consider the comparison between ACLs A_{n-1} and A_n. For each non-overlapping rule nr from A_{n-1}, let $V_{F_l,h}(nr)$ denote the h-th ($1 \leq h \leq 3$) prefix set for the field F_l ($1 \leq l \leq d$), *e.g.*, in Figure 5.8(a), $V_{F_l,1}(nr_1^{(2)})$ denotes {00010, 00101}. Let $\mathcal{N}(\mathcal{F}(a_l))$ denote one set of prefixes for the field F_l, *e.g.*, $\mathcal{N}(\mathcal{F}(0))$ in Figure 5.8(c). The basic operation of the third phase is to compare whether two sets from different ACLs, *e.g.*, $V_{F_l,h}(nr)$ and $\mathcal{N}(\mathcal{F}(a_l))$, have a common element. According to the theorems in multi-party secure computation [9, 34] and the theorem in [21], we can prove that after the three phases, the party P_{n-1} only learns $V_{F_l,h}(nr) \cap \mathcal{N}(\mathcal{F}(a_l))$ and the size of $\mathcal{N}(\mathcal{F}(a_l))$.

To prove this claim, based on the theorems of multi-party secure computation [9, 34], we only need to prove that the distribution of the P_{n-1}'s view of our protocol cannot be distinguished from a simulation that uses only $V_{F_l,h}(nr)$, $V_{F_l,h}(nr) \cap \mathcal{N}(\mathcal{F}(a_l))$, and the size of $\mathcal{N}(\mathcal{F}(a_l))$. The theorem in [21] proves that P_{n-1}'s view of our protocol

$$Y_R = \{\underbrace{(x_1)_{K_{n-1}}, \cdots, (x_m)_{K_{n-1}}}_{x_i \in V_{F_l,h}(nr) \cap \mathcal{N}(\mathcal{F}(a_l))}, \underbrace{(x_{m+1})_{K_{n-1}}, \cdots, (x_t)_{K_{n-1}}}_{x_i \in V_{F_l,h}(nr) - \mathcal{N}(\mathcal{F}(a_l))}\}$$

cannot be distinguished from the simulation

$$Y_S = \{\underbrace{(x_1)_{K_{n-1}}, \cdots, (x_m)_{K_{n-1}}}_{x_i \in V_{F_l,h}(nr) \cap \mathcal{N}(\mathcal{F}(a_l))}, \underbrace{z_{m+1}, \cdots, z_t}_{t-m=|V_{F_l,h}(nr) - \mathcal{N}(\mathcal{F}(a_l))|}\}$$

where $z_{m+1}, \cdots, z_t$ are random values and uniformly distributed in the domain of encrypted numbers.

Knowing $V_{F_l,h}(nr) \cap \mathcal{N}(\mathcal{F}(a_l))$ and the size of $\mathcal{N}(\mathcal{F}(a_l))$, P_{n-1} cannot reveal the rules in A_n for two reasons. First, a numericalized prefix in $V_{F_l,h}(nr) \cap \mathcal{N}(\mathcal{F}(a_l))$ can be generated from many numbers. Considering a numericalized prefix of 32-bit IP addresses $b_1 b_2 \cdots b_k 10 \cdots 0$, the number of possible IP addresses that can generate such prefix is 2^{32-k}. Furthermore, after the comparison, P_{n-1} sends to P_n the comparison result which is encrypted with

P_{n-1}'s secret key K_{n-1}. Without knowing K_{n-1}, P_n cannot reveal the comparison result, and hence, cannot reveal the values from A_{n-1}. Second, the size of $\mathcal{N}(\mathcal{F}(a_l))$ cannot be used to reveal the rules in A_n because for any a_l or b_l in the field F_l of A_n, the size of $\mathcal{N}(\mathcal{F}(a_l))$ or $\mathcal{N}(\mathcal{F}(b_l))$ is constant.

At the end of our protocol, only P_1 knows the intersection of n ACLs, which includes some information (*i.e.*, numbers) from other ACLs. However, our goal is to preserve the privacy of ACLs, not the privacy of the intersection result. Knowing such numbers cannot help P_1 to reveal an ACL rule of other parties for two reasons. First, a real ACL typically consists of hundreds of rules and no one consists of only one rule. Second, P_1 does not know which numbers belong to A_j $(2 \leq j \leq n)$, which two numbers form an interval, and which d intervals form a rule in A_j. The number of possible combinations can be extremely large. Considering the intersection in Figure 5.9, P_1 cannot know which ACL, A_2 or A_3, contains the number 2 or the number 6.

5.4.2 Complexity Analysis

In this section, we analyze the computation, space, and communication costs in our protocol. Let m_i be the number of rules in ACL A_i $(1 \leq i \leq n)$ and d be the number of fields. For ease of presentation, assume that different fields have the same length, *i.e.*, w bits. We first analyze the complexity of processing ACLs $A_1, \cdots, A_{n-1}$ and then analyze the complexity of processing ACL A_n. The maximum number of non-overlapping rules generated from the FDD is $(2m_i-1)^d$ [49]. Each non-overlapping rule consists of d w-bit intervals, each interval can be converted to at most three ranges, and each range can be converted to at most $2w-2$ prefixes [39]. Thus, the maximum number of prefixes generated from these non-overlapping rules is $3d(2w-2)(2m_i-1)^d$. Recall that P_i $(1 \leq i \leq n-1)$ unions all prefixes into one set. Then, the number of prefixes cannot exceed 2^{w+1}. There-

fore, for processing A_i $(1 \leq i \leq n-1)$, the computation cost of encryption by P_i, $\cdots$, P_n is $\min(3d(2w-2)(2m_i-1)^d, 2^{w+1}) = \min(O(dwm_i^d), O(2^w))$, the space cost of P_i is $O(dwm_i^d)$, and the communication cost is $\min(O(dwm_i^d), O(2^w))$. For processing P_n, each interval of the non-overlapping rules is converted to two prefix families and each prefix family includes $w+1$ prefixes. Thus, the maximum number of prefixes converted from A_n is $2d(w+1)(2m_n-1)^d$. Therefore, for processing A_n, the computation, space, and communication costs of P_n is $O(dwm_n^d)$.

5.5 Optimization

To reduce the computation and communication costs, we divide the problem of computing reachability of n ACLs to the problem of computing reachability of two ACLs. Then the intermediate results are aggregated hierarchically to obtain final reachability result. Let Q_i $(1 \leq i \leq n)$ denote the set of non-overlapping accepting rules from ACL A_i. In the ACL encoding and encryption phase, Q_i is encrypted $n-i+1$ times, *i.e.*, encrypted by P_i, P_{i+1}, $\cdots$, P_n. Thus, the number of encryptions for $Q_1, \cdots, Q_n$ is $n + (n-1) + \cdots + 1 = O(n^2)$. Similarly, the number of messages in this phase is $O(n^2)$. To reduce the the number of encryptions and messages, we first divide n ACLs into $\lfloor n/2 \rfloor$ groups. The j-th $(1 \leq j \leq \lfloor n/2 \rfloor)$ group includes two adjacent ACLs A_{2j-1} and A_{2j}. The last group includes adjacent ACLs $A_{2\lfloor n/2 \rfloor - 1}, \cdots, A_n$. For example, 5 ACLs can be divided into two groups $\{A_1, A_2\}$ and $\{A_3, A_4, A_5\}$. Second, for the ACLs in each group, we run the proposed protocol to compute the network reachability. The result for each group is actually a new set of non-overlapping accepting rules. Therefore, we obtain $\lfloor n/2 \rfloor$ sets of non-overlapping accepting rules. Repeat these two steps until we obtain the network reachability for all n ACLs. Through this process, there are $\lfloor n/2 \rfloor + \lfloor n/2^2 \rfloor + ... + 1 = O(n)$ groups, and for each group with two ACLs, the number of ACL encryptions and messages is $2+1=3$. Thus, the number of ACL

encryptions and messages is reduced from $O(n^2)$ to $O(n)$.

5.6 Experimental Results

We evaluated the efficiency and effectiveness of our protocol on 10 real ACLs and 100 synthetic ACLs. Both real and synthetic ACLs examine five fields, source IP, destination IP, source port, destination port, and protocol type. For real ACLs, the number of rules ranges from hundreds to thousands, and the average number of rules is 806. Due to security concerns, it is difficult to obtain many real ACLs. Thus, we generated a large number of synthetic ACLs based on Singh *et al.* 's technique [74]. For synthetic ACLs, the number of rules ranges from 200 to 2000, and for each number, we generated 10 synthetic ACLs. In implementing the commutative encryption, we used the Pohlig-Hellman algorithm [64] with a 1024-bit prime modulus and 160-bit encryption keys. Our experiments were implemented using Java 1.6.0.

To evaluate the effectiveness, we verified the correctness of our protocol because we knew all the ACLs in the experiments. The results show that our protocol is deterministic and accurate with the given ACLs. Thus, in this section, we focus on the efficiency of our protocol. Recall that processing ACL A_i ($1 \leq i \leq n-1$) is different from processing the last destination ACL A_n. Therefore, we evaluate the computation and communication costs of the core operations of our protocol, processing ACL A_i ($1 \leq i \leq n-1$), processing the destination ACL A_n, and comparing A_i and A_n. Knowing this performance, we can easily estimate time and space consumption for a given network path with n ACLs belonging to n parties.

5.6.1 Efficiency on Real ACLs

Our protocol is efficient for processing real ACL A_i ($1 \leq i \leq n-1$). Figure 5.10(a) shows for processing A_i the computation cost of P_i and the average computation cost of other parties

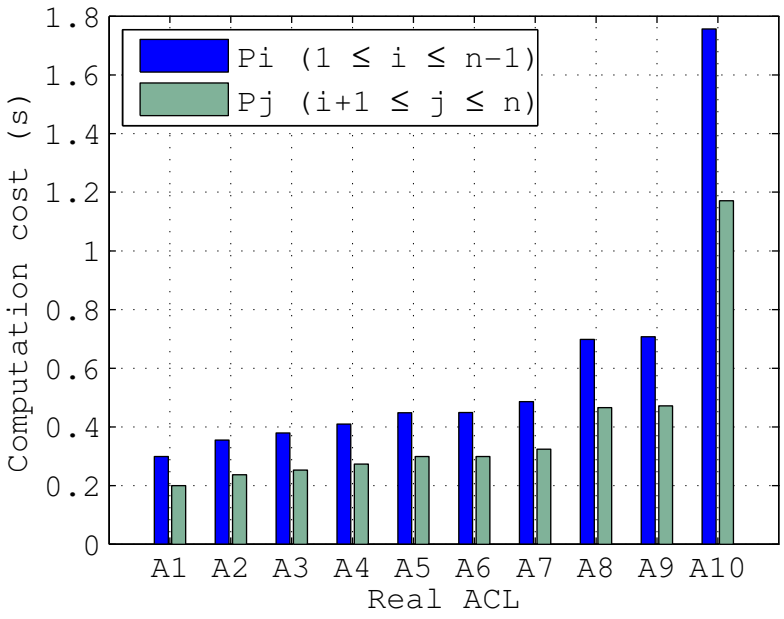
$P_{i+1}, \cdots, P_n$. The computation cost of P_i is less than 2 seconds and the computation cost of P_j $(i+1 \leq j \leq n)$ is less than 1.5 seconds. Note that, for processing A_i, the computation cost of P_i is one-time offline cost because P_i knows A_i, while the computation cost of P_j $(i+1 \leq j \leq n)$ is online cost. Figure 5.10(b) shows the average communication cost between any two adjacent parties P_j and P_{j+1} $(i \leq j \leq n)$ for processing ACL A_i, which is less than 60 KB. Note that, the computation costs of different parties P_j $(i+1 \leq j \leq n)$ are similar because they encrypt the same number of prefixes from A_i. Hence, we only show the average computation cost of parties $P_{i+1}, \cdots, P_n$. Similarly, the communication costs between every two adjacent parties P_j and P_{j+1} are the same.

Our protocol is efficient for processing real ACL A_n. Figure 5.11(a) shows that for processing A_n, the computation cost of P_n and the average computation cost of other parties. The computation cost of P_n is less than 10 seconds. The average computation cost of other parties is less than 6 seconds. Similarly, for processing A_n, the computation cost of P_n is one-time offline cost, while the computation costs of other party is online cost. Figure 5.11(b) shows the average communication cost between P_n and P_i $(1 \leq i \leq n-1)$, which is less than 410 KB.

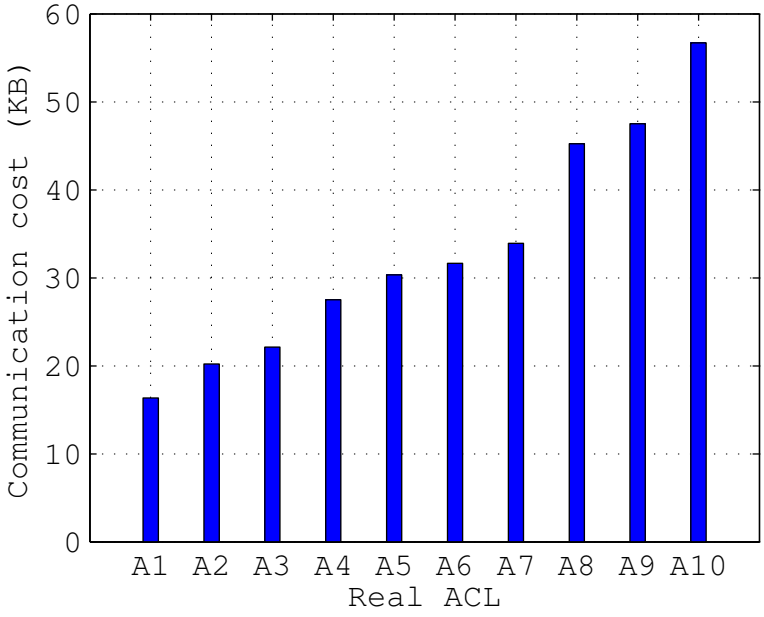
Our protocol is efficient for real ACL comparison. The comparison time between two ACLs is less than 1 second, which is much less than the computation cost of processing ACLs. Because the commutative encryption is more expensive than checking whether two sets have a common element.

5.6.2 Efficiency on Synthetic ACLs

To further evaluate the efficiency, we executed our protocol over every 10 synthetic ACLs with the same number of rules, and then measured the computation and communication costs of operations on synthetic ACLs A_1 to A_n for different parties. Particularly, we measured

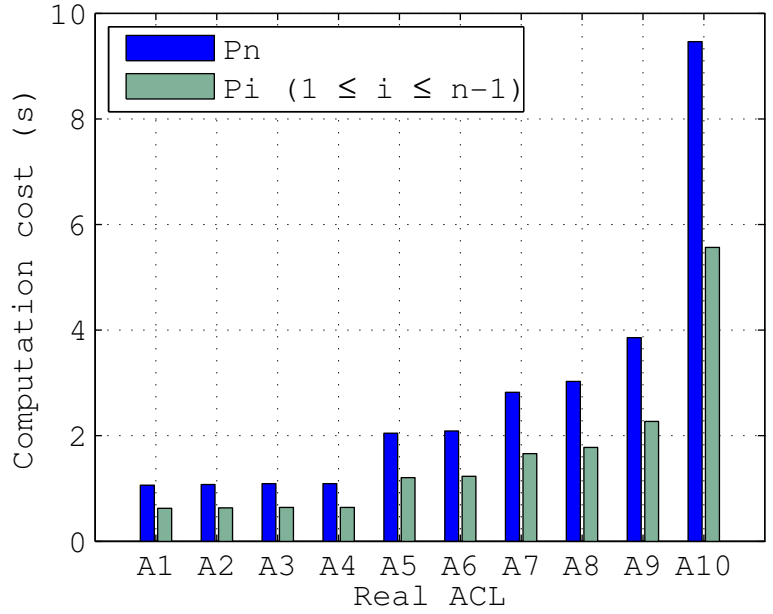


(a) Computation cost

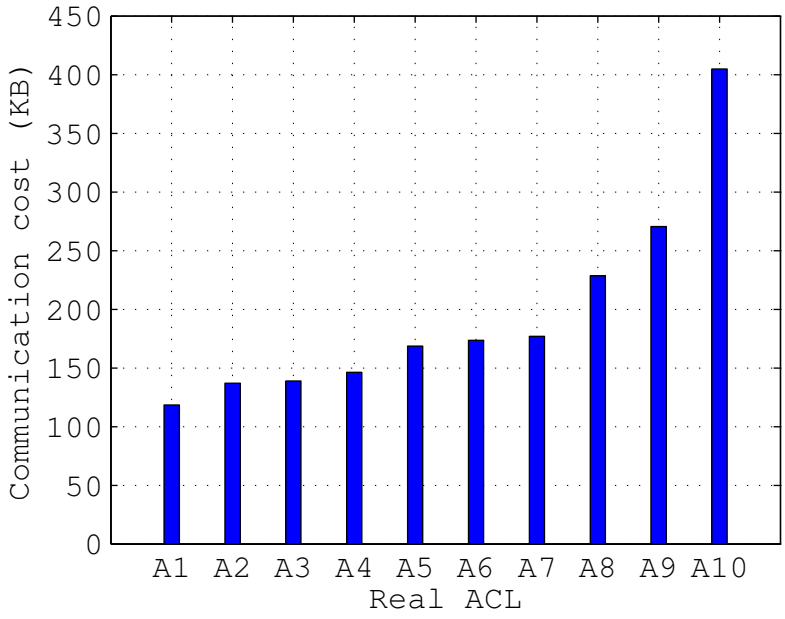


(b) Communication cost

Figure 5.10. Comp. & comm. costs for processing real ACL A_i ($1 \leq i \leq n-1$)



(a) Computation cost



(b) Communication cost

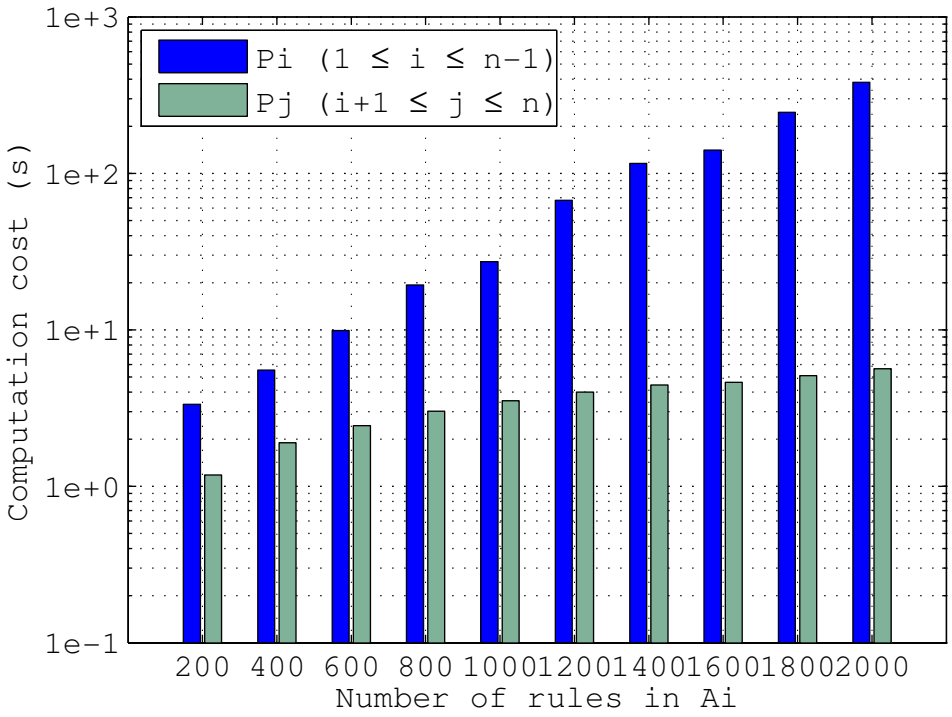
Figure 5.11. Comp. & comm. costs for processing real ACL A_n

computation and communication costs for processing each synthetic ACL A_i ($1 \leq i \leq n-1$), processing synthetic ACL A_n, and the comparison time for every two ACLs.

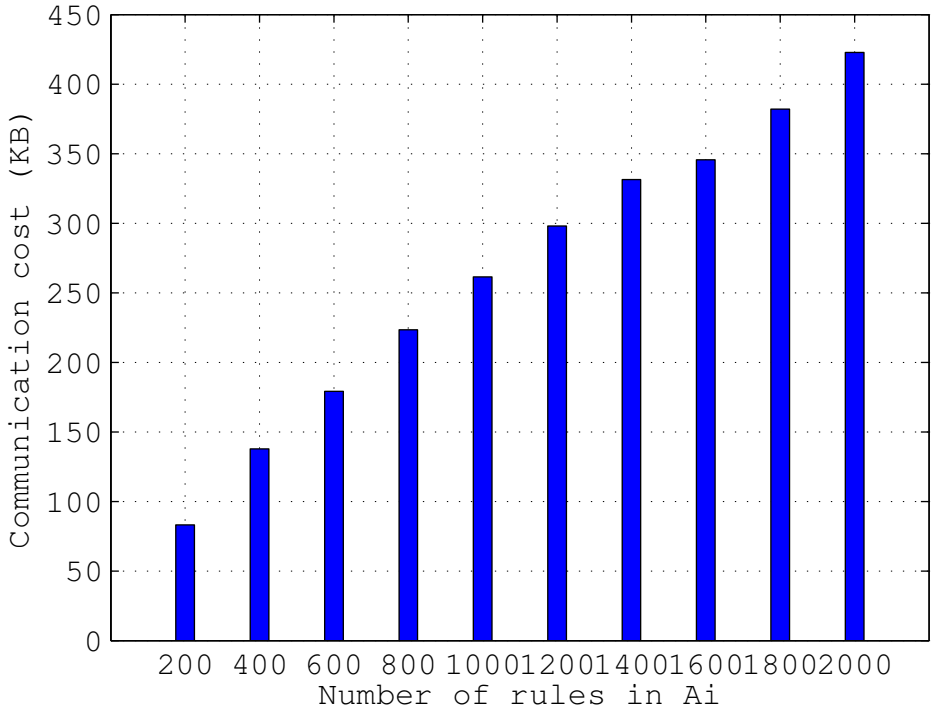
For processing synthetic ACL A_i ($1 \leq i \leq n-1$), Figure 5.12(a) shows the computation cost of P_i and the average computation cost of parties $P_{i+1}, \cdots, P_n$ and Figure 5.12(b) shows the average communication cost between P_j and P_{j+1} ($i \leq j \leq n$). The one-time offline computation cost (*i.e.*, the computation cost of P_i) is less than 400 seconds, and the online computation cost (*i.e.*, the average computation cost of other parties P_j) is less than 5 second. The average communication cost between any two adjacent parties P_j and P_{j+1} is less than 450 KB.

For processing synthetic ACL A_n, Figure 5.13(a) shows the computation cost of P_n and the average computation cost of other parties and Figure 5.13(b) shows the average communication cost between P_n and P_i ($1 \leq i \leq n-1$). The one-time offline computation cost (*i.e.*, the computation cost of P_n) is less than 550 seconds, and the online computation cost (*i.e.*, the average computation cost of other parties $P_1, \cdots, P_{n-1}$) is less than 25 seconds. The average communication cost between P_n and P_i is less than 2100 KB.

Figure 5.14 shows the average comparison time for every two synthetic ACLs. The comparison time between any two synthetic ACLs is less than 5 seconds, which is much more efficient than processing ACLs.

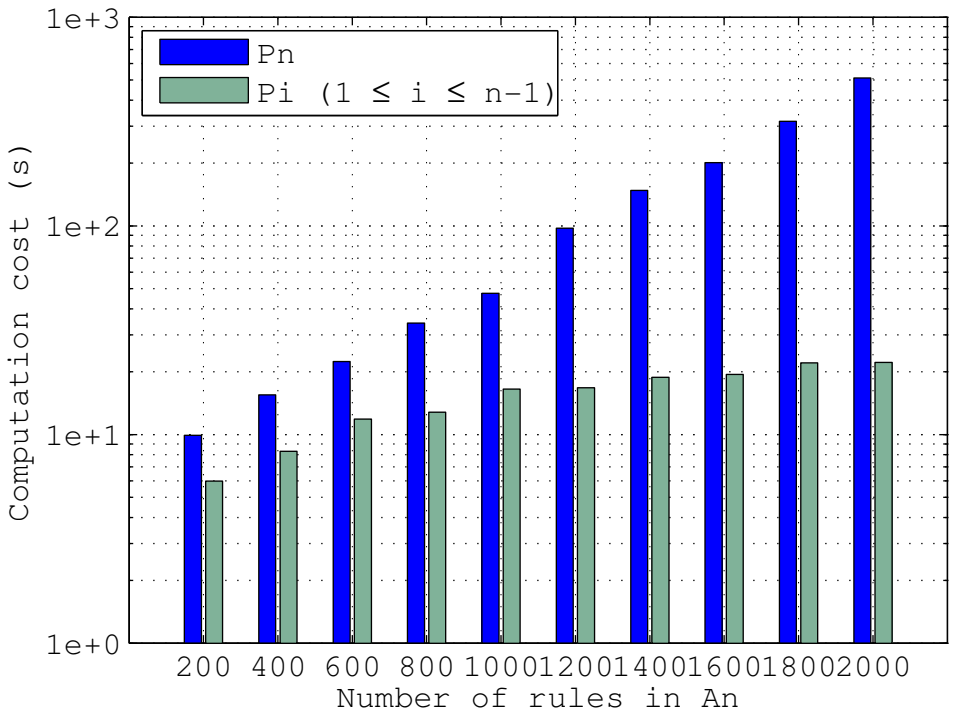


(a) Ave. computation cost

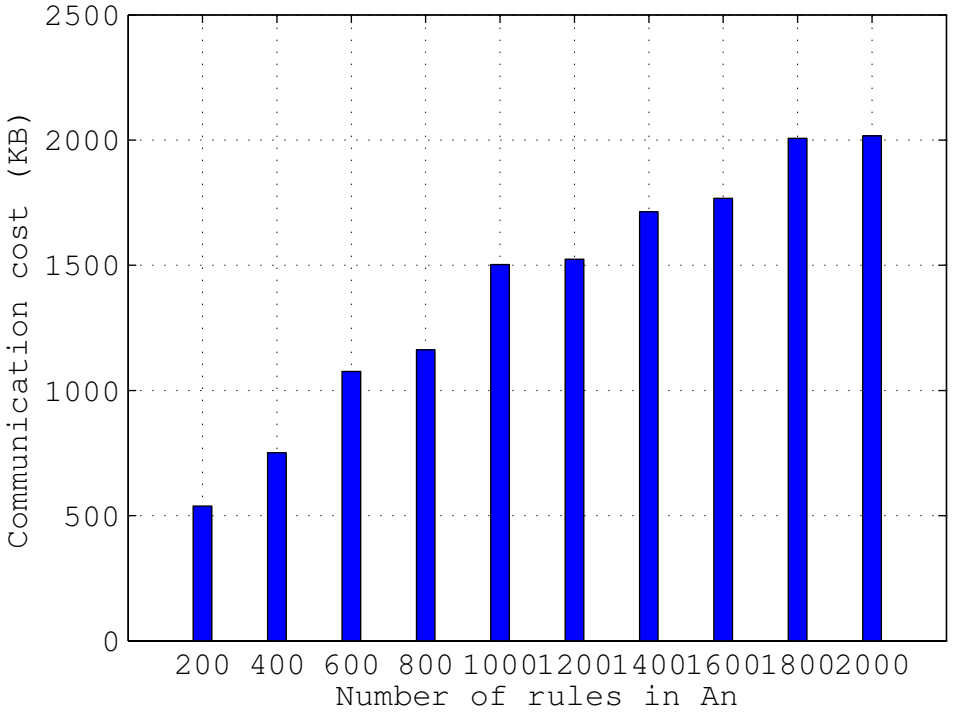


(b) Ave. communication cost

Figure 5.12. Comp. & comm. costs for processing synthetic ACL A_i ($1 \leq i \leq n-1$)



(a) Ave. computation cost



(b) Ave. communication cost

Figure 5.13. Comp. & comm. costs for processing synthetic ACL A_n

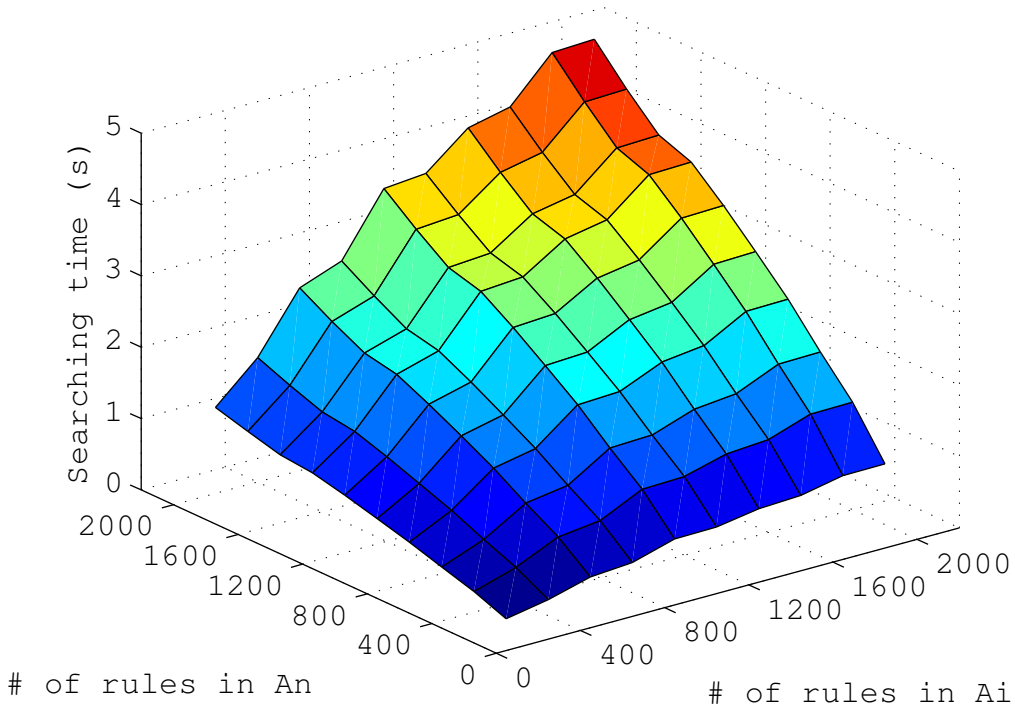


Figure 5.14. Comparison time of synthetic ACLs A_i and A_n

CHAPTER 6

Related Work

6.1 Secure Multiparty Computation

One of the fundamental cryptographic primitives for designing privacy-preserving protocols is secure multiparty computation, which was first introduced by Yao with the famous "Two-Millionaire Problem" [79]. A secure function evaluation protocol enables two parties, one with input x and the other with y, to cooperatively compute a function $f(x, y)$ without disclosing one party's input to the other. The classical solutions are Yao's "garbled circuits" protocol [80] and Goldreich's protocol [60]. Other related work is privacy preserving set operations, which enables n parties, each party with its private set s_i, to collaboratively compute the intersection of all sets, $s_1 \cap \cdots \cap s_n$, without disclosing more information of one party's private set beyond the intersection to other parties [33, 45, 68]. Although we could apply these solutions to solve the problem of privacy preserving network reachability, the computation and communication costs of these solutions is prohibitive due to the unnecessary requirement of secure multiparty computation for our problems. Secure multiparty computation requires every party to know the result, while in our problems only one party knows the result. This difference significantly affects computation and communication costs.

6.2 Privacy and Integrity Preserving in WSNs

Privacy and integrity preserving range queries in wireless sensor networks (WSNs) have drawn people's attention recently [69, 73, 84]. Sheng and Li proposed a scheme to preserve the privacy and integrity of range queries in sensor networks [69]. This scheme uses the bucket partitioning idea proposed by Hacigumus *et al.* in [40] for database privacy. The basic idea is to divide the domain of data values into multiple buckets, the size of which is computed based on the distribution of data values and the location of sensors. In each time slot, a sensor collects data items from the environment, places them into buckets, encrypts them together in each bucket, and then sends each encrypted bucket along with its bucket ID to a nearby storage node. For each bucket that has no data items, the sensor sends an encoding number, which can be used by the sink to verify that the bucket is empty, to a nearby storage node. When the sink wants to perform a range query, it finds the smallest set of bucket IDs that contains the range in the query, then sends the set as the query to storage nodes. Upon receiving the bucket IDs, the storage node returns the corresponding encrypted data in all those buckets. The sink can then decrypt the encrypted buckets and verify the integrity using encoding numbers. S&L scheme only considered one-dimensional data in [69] and it can be extended to handle multi-dimensional data by dividing the domain of each dimension into multiple buckets.

S&L scheme has two main drawbacks, which are inherited from the bucket partitioning technique. First, as pointed out in [41], the bucket partitioning technique allows compromised storage nodes to obtain a reasonable estimation on the actual value of both data items and queries. In comparison, in SafeQ, such estimations are very difficult. Second, for multi-dimensional data, the power consumption of both sensors and storage nodes, as well as the space consumption of storage nodes, increases exponentially with the number of dimensions due to the exponential increase of the number of buckets. In comparison, in SafeQ, the power

and space consumption increases linearly with the number of dimensions times the number of data items.

Shi *et al.* proposed an optimized version of S&L's integrity preserving scheme aiming to reduce the communication cost between sensors and storage nodes [73, 84]. The basic idea of their optimization is that each sensor uses a bit map to represent which buckets have data and broadcasts its bit map to the nearby sensors. Each sensor attaches the bit maps received from others to its own data items and encrypts them together. The sink verifies query result integrity for a sensor by examining the bit maps from its nearby sensors. In our experiments, we did not choose the solutions in [73, 84] for side-by-side comparison for two reasons. First, the techniques used in [73, 84] are similar to S&L scheme except the optimization for integrity verification. The way they extend S&L scheme to handle multi-dimensional data is to divide the domain of each dimension into multiple buckets. They inherit the same weakness of allowing compromised storage nodes to estimate the values of data items and queries with S&L scheme. Second, their optimization technique allows a compromised sensor to easily compromise the integrity verification functionality of the network by sending falsified bit maps to sensors and storage nodes. In contrast, in S&L and our schemes, a compromised sensor cannot jeopardize the querying and verification of data collected by other sensors.

6.3 Privacy and Integrity Preserving in DAS

Database privacy has been studied extensively in both database and security communities (*e.g.*, [40, 41, 10, 16, 31, 72, 17]). Based on whether query results include false positives, *i.e.*, data items that do not satisfy the queries but are included in the query results, we can classify these schemes into the following two categories: approximate privacy-preserving schemes [40, 41] and precise privacy-preserving schemes [10, 16, 31, 72, 17]. The *approximate*

privacy-preserving schemes reply query results with false positives but they are more efficient, while the *precise privacy-preserving schemes* reply query results without false positives but they are more expensive. Next, we discuss these two categories of privacy-preserving schemes.

Approximate Privacy-Preserving Schemes

Hacigumus *et al.* first proposed the bucket partitioning idea for querying encrypted data in the database-as-service model (DAS) [40]. The basic idea is to divide the attribute domains into multiple buckets and then map bucket ids to random numbers for preserving privacy. Later, Hore *et al.* explored the optimal partitioning of buckets [41]. However, as pointed out in [41], bucket partitioning incurs a tradeoff between privacy and efficiency. If the bucket sizes are large, less privacy information is leaked, but query results include more false positives; if the bucket sizes are small, more privacy information is leaked, but query results include less false positives. In contrast, our scheme is a precise privacy-preserving scheme, which returns query results without false positives. Furthermore, it leaks only the minimum privacy information for any possible precise privacy-preserving scheme, which will be discussed in Section 3.3.4.

Precise Privacy-Preserving Schemes

Previous work on order-preserving hash functions [31] and order-preserving encryptions [10, 16] can be employed for constructing precise privacy-preserving schemes in cloud computing. Fox *et al.* proposed an order-preserving minimal prefect hash function (OPMPHF) for a domain with N possible values [31]. Agrawal *et al.* proposed an order-preserving encryption (OPES) [10]. The basic idea of OPES is to transform data items to different values such that the transformed values preserve the order of the data items without disclosing the privacy of the data items to cloud providers. Specifically, this scheme first divides the data domain into multiple buckets, *i.e.*, m buckets, computes a transformation polynomial function for each bucket, and then applies the corresponding transformation function to the

data items in each bucket. However, for z-dimensional data items, the OPMPHF function requires $O(zN \log N)$ shared secret information between the organization and its customers, and the OPES encryption requires $O(zm)$ shared secret information. In contrast, our order-preserving hash-based function only requires $O(z)$ shared secret information.

Boneh & Waters proposed a public-key system for supporting conjunctive, subset, and range queries on encrypted data [17]. Although theoretically it seems possible, Boneh&Waters' scheme cannot be used to solve the privacy problem in our context because it is too computationally expensive for cloud computing. It would require an organization to perform $O(zN)$ encryption for submitting data to a cloud provider, where z is the number of dimensions and N is the domain size (*i.e.*, the number of all possible values) of each dimension. Here N could be large and each encryption is expensive due to the use of public-key cryptography. Shi *et al.* proposed another public-key system for supporting multi-dimensional range queries on encrypted data [72]. However, this scheme has two drawbacks in cloud computing. First, it is not practical to require an organization to stay online after it outsourced the data to cloud providers. In Shi *et al.* 's scheme, for each distinct range query, a customer needs to retrieve a different decryption key from the organization. Second, it is not efficient to process queries. In order to process a query, a cloud provider needs to scan all the encrypted data from an organization. In contrast, our privacy-preserving scheme not only can be computed efficiently due to the use of the hash function and symmetric encryption, but also enables cloud providers to perform binary searches to process queries.

Database integrity has also been explored in prior work [27, 63, 62, 59, 24, 22], independent of database privacy. Merkle hash trees have been used for the authentication of data items [58] and they were used for verifying the integrity of range queries in [27, 63]. However, it is difficult to extend Merkle hash trees for supporting multi-dimensional range queries. Pang *et al.* [62] and Narasimha & Tsudik [59] proposed similar schemes for verifying the integrity of range query results using signature aggregation and chaining. For each data item, Pang

et al. computed the signature of the data item by signing the concatenation of the digests of the data item itself as well as its left and right neighbors [62]. Narasimha & Tsudik computed the signature by signing the concatenation of the digests of the data item and its left neighbors along each dimension [59]. However, signature aggregation and chaining requires a cloud provider to reply to the customer the boundary data items of the query that do not satisfy the query.

Chen *et al.* proposed Canonical Range Trees (CRTs) to store the counting information for multi-dimensional data such that the counting information can be used for integrity verification without leaking boundary data items of the query [22]. However, we argue that the most important requirement in cloud computing is to preserve data privacy. CRTs contain a lot of privacy information. Thus, we need to preserve the privacy of CRTs. As we discussed before, preserving privacy of relational databases is already difficult. Preserving privacy of CRT trees is much more difficult and no scheme has been proposed. Furthermore, their scheme requires an organization to send a multi-dimensional CRT with $O(n \log^z N)$ overhead to a cloud provider, where n is the number of data items and N is the domain size of each dimension. Therefore, it incurs too much communication cost between an organization and a cloud provider.

6.4 Firewall Redundancy Removal and Collaborative Firewall Enforcement in VPN

Prior work on intra-firewall redundancy removal aims to detect redundant rules within a single firewall [38, 50, 52]. Gupta identified backward and forward redundant rules in a firewall [38]. Later, Liu *et al.* pointed out that the redundant rules identified by Gupta are incomplete, and proposed two methods for detecting all redundant rules [50, 52]. Prior

work on inter-firewall redundancy removal requires the knowledge of two firewall policies and therefore is only applicable within one administrative domain [11, 81].

Prior work on collaborative firewall enforcement in virtual private networks (VPNs) enforces firewall policies over encrypted VPN tunnels without leaking the privacy of the remote network's policy [23, 48]. The problems of collaborative firewall enforcement in VPNs and privacy-preserving inter-firewall optimization are fundamentally different. First, their purposes are different. The former focuses on enforcing a firewall policy over VPN tunnels in a privacy-preserving manner, whereas the latter focuses on removing inter-firewall redundant rules without disclosing their policies to each other. Second, their requirements are different. The former preserves the privacy of the remote network's policy, whereas the latter preserves the privacy of both policies.

6.5 Network Reachability Quantification

The challenges in network reachability include, misconfiguration of ACLs, changes of routing policies, and link failures, that could prevent accessibility to essential network services. To estimate reachability, existing approaches analyze ACLs while considering other critical parameters like dynamic routing policies, packet transforms, and variations in protocol operations [12, 42, 44, 54, 75, 77]. To estimate the bounds on reachability, Xie *et al.* defined union and intersection operations over ACLs while taking into account the routing decisions, packet transforms, and link failures [77]. This approach, however, over approximates and does not yield exact bounds. Ingols *et al.* used Binary Decision Diagrams (BDDs) to reduce the complexity of handling ACLs and to estimate reachability more accurately [42]. Matousek *et al.* described a formal model using Interval Decision Diagrams (IDDs) to analyze network reachability under all possible network failure conditions [54]. However, the approach is not scalable as it performs an exhaustive evaluation of failure scenarios that may

or may not occur. Al-Shaer *et al.* proposed a more accurate model using BDDs and applied symbolic model checking techniques on properties specified in computation tree logic (CTL) [30], to verify reachability across the network for any given packet [12]. Sung *et al.* studied the effect of reachability constraints on class-of-service flows, where the packets are subjected to an additional constraint based on their class-of-service [75]. Khakpour *et al.* used Firewall Decision Diagrams (FDDs) to quantify reachability while considering all possible network transforms like NAT, PAT as well as protocol states like connection-oriented, state-less and so on [44]. They also described a query language to enable network operators to execute reachability queries.

While most solutions operate on static configurations, some work has been proposed for estimating reachability in an online manner. Bandhakavi *et al.* analyzed the network reachability using a simulated routing environment, *i.e.*, they constructed a routing graph which represents the possible paths that could be taken by routing advertisements under the current router configurations [13]. Analyzing the graph can identify violations in security policies and in verifying reachability. Zhang *et al.* described a real-time monitoring and verification system for network reachability [83]. A monitoring software runs on all the routers and collects up-to-date ACL and forwarding state information, which enables the network administrator to determine instantaneous reachability between any source destination pair. This approach provides an insight into instantaneous reachability as it considers snapshots of routing tables at the time of analysis.

Several other works have been proposed to reduce the complexity of managing networks and to verify network configurations. Casado *et al.* described a novel architecture for enterprise, Secure Architecture for the Networked Enterprise (SANE), which comprises of a centralized authentication server that allows authorized users to access services [19]. In SANE, the ACLs can be specified in a natural way so as to capture the semantics clearly. Le *et al.* used data mining techniques to analyze security policies and to detect possible

misconfigurations in the policies [47]. They considered the notion of association rule mining to extract usable safe configurations of routers and detect anomalies in other routers using the extracted patterns. Benson *et al.* described complexity metrics to evaluate relative complexity among alternate network designs [14]. The metrics allow network operators to compare configurations with standard configurations and identify errors.

All these approaches are based on the same assumption, that is, there is a central network analyst who has the complete knowledge of the network configuration and other critical information. However, this assumption is not true for a network where network devices belong to different parties whose network configuration cannot be shared with other parties. Therefore, these approaches cannot quantify network reachability across different parties.

CHAPTER 7

Conclusions and Future Work

Preserving privacy and integrity of private data has become core requirements in the recent decade for distributed systems across different parties. In this dissertation, we investigate four important privacy and integrity preserving problems for different distributed systems.

For two-tiered sensor networks, we propose SafeQ, a effective and efficient protocol for handling range queries in a privacy and integrity preserving fashion. SafeQ uses the techniques of prefix membership verification, Merkle hash trees, and neighborhood chaining. In terms of security, SafeQ significantly strengthens the security of two-tiered sensor networks. Unlike prior art, SafeQ prevents a compromised storage node from obtaining a reasonable estimation on the actual values of sensor collected data items and sink issued queries. In terms of efficiency, our results show that SafeQ significantly outperforms prior art for multi-dimensional data in terms of both power consumption and storage space. We also propose an optimization technique using Bloom filters to significantly reduce the communication cost between sensors and storage nodes.

For cloud computing, we propose novel privacy and integrity preserving schemes for multi-dimensional range queries. To preserve privacy, we propose an order-preserving hash-based function to encode the data from an organization and the queries from its customers such that

a cloud provider can process encoded queries over encoded data without knowing the actual values. To preserve integrity, we propose the first probabilistic integrity-preserving scheme for range queries. This scheme employs a new data structure, local bit matrices, which allows customers to verify the integrity of query results with high probability. Our future work will consider database updating and optimal bucket partitioning for multi-dimensional data, which have not been solved in this paper.

For distributed firewall policies, we identify an important problem, cross-domain privacy-preserving inter-firewall redundancy detection and propose a novel privacy-preserving protocol for detecting such redundancy. The results on real firewall policies show that our protocol can remove as many as 49% of the rules in a firewall whereas the average is 19.4%.

For network reachability, we address the problem of privacy preserving quantification of network reachability across different domains. Protecting the privacy of access control configuration is important as the information can be easily abused. We propose an efficient and secure protocol to quantify the network reachability accurately while protecting the privacy of ACLs. We use the divide-and-conquer strategy to decompose the reachability computation which results in a magnitude reduction of the computation and communication costs. Our future work will consider dynamic routing information and topological variations where links go down or new links get added to the network resulting in new paths for data propagation. In our protocol, we have considered the routing information partially by taking snapshots of the routing state and encoding it as an ACL rule. However, for a more fine-grained analysis, the instantaneous forwarding information state along with the local routing policies and other service level agreements also need to be considered.

APPENDICES

A Analysis of SafeQ Optimization

Let $HMAC_g(\mathcal{N}(\mathcal{S}([d_0, d_1]))), \cdots, HMAC_g(\mathcal{N}(\mathcal{S}([d_n, d_{n+1}])))$ be the sets of data items that a sensor needs to represent in a Bloom filter. Let $[a, b]$ be the range query over the n data items $d_1, \cdots, d_n$. Let w_o denote the bit length of each d_j, a, and b. Let w_h denote the bit length of the numbers after hashing in each $HMAC_g(\mathcal{N}(\mathcal{S}([d_j, d_{j+1}])))$. Let k be the number of hash functions in the Bloom filter.

Given two arrays A and B representing data in $HMAC_g(\mathcal{N}(\mathcal{S}([d_0, d_1])))$, $\cdots$, $HMAC_g(\mathcal{N}(\mathcal{S}([d_n, d_{n+1}])))$, for any v of w_h bits, a storage node searches the corresponding index for v by applying the k hash functions to v and check whether two conditions hold: (1) for every $1 \leq i \leq k$, $A[h_i(v)] = 1$; (2) for every $1 \leq i \leq k$, index j $(0 \leq j \leq n)$ is included in the list that $B[h_i(v)]$ points to. Let $\mathcal{X}(v)$ denote the index that the storage node finds for v: if the index exists (*i.e.*, the above conditions hold), $\mathcal{X}(v) = j$; otherwise, $\mathcal{X}(v) = null$.

Based on the analysis of Bloom filters [15], the probability $Pr(A[h_1(v)] = 1, \cdots, A[h_k(v)] = 1)$ is $(1 - e^{-k(n+1)q/c})^k$. The probability $Pr(j \in B[h_1(v)], \cdots, j \in B[h_k(v)])$ is $(\frac{1}{n+1})^k$. Therefore, we have

$$Pr(\mathcal{X}(v) = j) = (1 - e^{-k(n+1)q/c})^k (\frac{1}{n+1})^k \tag{7.1}$$

As $Pr(\mathcal{X}(v) = j)$ is the same for any $0 \leq j \leq n$, let α denote the probability $Pr(\mathcal{X}(v) = j)$. According to our discussion in Section 2.3.1, each of the two sets $HMAC_g(\mathcal{N}(\mathcal{F}(a)))$ and $HMAC_g(\mathcal{N}(\mathcal{F}(b)))$ includes $w_o + 1$ w_h-bit numbers. For $HMAC_g(\mathcal{N}(\mathcal{F}(a)))$, there exists a range $[d_{n_1-1}, d_{n_1}]$ such that $a \in [d_{n_1-1}, d_{n_1}]$. Therefore, there exists one number v_a in $HMAC_g(\mathcal{N}(\mathcal{F}(a)))$ such that $\mathcal{X}(v_a) = n_1 - 1$. Let $v_1, \cdots, v_{w_o}$ denote the rest w_o numbers in $HMAC_g(\mathcal{N}(\mathcal{F}(a)))$ and $\mathcal{Y}$ denote the *minimum* index in $\{\mathcal{X}(v_1), \cdots, \mathcal{X}(v_{w_o})\}$. Without loss of generality, we assume $\mathcal{X}(v_1)$ is the minimum index. The probability of $\mathcal{Y} = j_1 - 1$

can be computed as follows:

$$\begin{aligned} Pr(\mathcal{Y} = j_1 - 1) &= Pr(\mathcal{X}(v_1) = j_1 - 1) \prod_{i=2}^{w_o} Pr(\mathcal{X}(v_i) \geq j_1 - 1 \text{ or } \mathcal{X}(v_i) = null) \\ &= Pr(\mathcal{X}(v_1) = j_1 - 1) \prod_{i=2}^{w_o} (1 - Pr(\mathcal{X}(v_i) < j_1 - 1)) \\ &= \alpha[1 - (j_1 - 1)\alpha]^{w_o - 1} \end{aligned}$$

Similarly, for $HMAC_g(\mathcal{N}(\mathcal{F}(b)))$, there exists a range $[d_{n_2-1}, d_{n_2}]$ such that $b \in [d_{n_2-1}, d_{n_2}]$. Therefore, there exists one number v_b in $HMAC_g(\mathcal{N}(\mathcal{F}(b)))$ such that $\mathcal{X}(v_b) = n_2 - 1$. Let $v_1, \cdots, v_{w_o}$ denote the rest w_o numbers in $HMAC_g(\mathcal{N}(\mathcal{F}(b)))$ and $\mathcal{Z}$ denote the *maximum* index in $\{\mathcal{X}(v_1), \cdots, \mathcal{X}(v_{w_o})\}$. Without loss of generality, we assume $\mathcal{X}(v_1)$ is the maximum index. We have

$$\begin{aligned} Pr(\mathcal{Z} = j_2 - 1) &= Pr(\mathcal{X}(v_1) = j_2 - 1) \prod_{i=2}^{w_o} Pr(\mathcal{X}(v_i) \leq j_2 - 1 \text{ or } \mathcal{X}(v_i) = null) \\ &= Pr(\mathcal{X}(v_1) = j_2 - 1) \prod_{i=2}^{w_o} (1 - Pr(\mathcal{X}(v_i) > j_2 - 1)) \\ &= \alpha[1 - (n - j_2 + 1)\alpha]^{w_o - 1} \end{aligned}$$

Given a query $\{HMAC_g(\mathcal{N}(\mathcal{F}(a))), HMAC_g(\mathcal{N}(\mathcal{F}(b)))\}$, if the storage node can find $\mathcal{Y} = j_1 - 1$ or $\mathcal{Z} = j_2 - 1$ where $0 \leq j_1 < n_1 \leq n_2 < j_2 \leq n$, the query result has false positives. Therefore, the average false positive rate can be computed as follows:

$$\begin{aligned} \epsilon = & \sum_{n_1=1}^{n+1} \sum_{n_2=n_1}^{n+1} \{ \sum_{j_1=1}^{n_1-1} \sum_{j_2=n_2+1}^{n+1} [\frac{(j_2 - j_1) - (n_2 - n_1)}{n - (n_2 - n_1)} \times \\ & Pr(\mathcal{Y} = j_1 - 1) Pr(\mathcal{Z} = j_2 - 1)] \\ & + \sum_{j_1=1}^{n_1-1} [\frac{n_1 - j_1}{n - (n_2 - n_1)} \times \\ & Pr(\mathcal{Y} = j_1 - 1) Pr(\mathcal{Z} < n_2 \text{ or } \mathcal{Z} = null)] \\ & + \sum_{j_2=n_2+1}^{n+1} [\frac{j_2 - n_2}{n - (n_2 - n_1)} \times \\ & Pr(\mathcal{Y} > n_1 - 2 \text{ or } \mathcal{Y} = null) Pr(\mathcal{Z} = j_2 - 1)]\} \end{aligned} \tag{7.2}$$

Because $[1-(j_1-1)\alpha]^{w_o-1} \leq 1$, $[1-(n-j_2+1)\alpha]^{w_o-1} \leq 1$, $\frac{n_1-j_1}{n-(n_2-n_1)} \leq 1$, $\frac{j_2-n_2}{n-(n_2-n_1)} \leq 1$, and $\frac{(j_2-j_1)-(n_2-n_1)}{n-(n_2-n_1)} \leq 1$, we derive Formula 2.1 from the following calculation.

$$\begin{aligned} \epsilon &< \sum_{n_1=1}^{n+1} \sum_{n_2=n_1}^{n+1} [n-(n_2-n_1)]\alpha \\ &= \frac{1}{3}\frac{(n+2)(n+3)}{(n+1)^{k-1}}(1-e^{-k(n+1)q/c})^k \end{aligned}$$

Typically, we choose the value $c = \frac{1}{\ln 2}k(n+1)q \approx 1.44k(n+1)q$ to minimize the probability of false positive for Bloom filters. Thus, Formula 2.1 becomes

$$\epsilon < \frac{1}{3}(\frac{1}{2})^k \frac{(n+2)(n+3)}{(n+1)^{k-1}}$$

Next, we discuss under what condition our optimization technique reduces the communication cost between sensors and storage nodes. To represent data in the $n+1$ sets $HMAC_g(\mathcal{N}(\mathcal{S}([d_0, d_1])))$, $\cdots$, $HMAC_g(\mathcal{N}(\mathcal{S}([d_n, d_{n+1}])))$, without Bloom filters, the total number of bits required is $w_h(n+1)q$; with Bloom filters, the total number of bits required is at most $c+2k(n+1)q\lceil \log_2 (n+1)\rceil$. Note that the number of bits for representing array A is c, the number of bits for representing array B is at most $2k(n+1)q\lceil \log_2 (n+1)\rceil$. Therefore, we derive Formula 2.2.

$$w_h(n+1)q > c + 2k(n+1)q\lceil \log_2 (n+1)\rceil$$

In case that $c = \frac{1}{\ln 2}k(n+1)q$, Formula 2.2 becomes

$$k \leq \frac{w_h}{\frac{1}{\ln 2} + 2\lceil \log_2 (n+1)\rceil} \approx \frac{w_h}{1.44 + 2\lceil \log_2 (n+1)\rceil}$$

B Properties of f_k^* and Their Proof

Order Preserving: *Assume any* $h_k(x_q) \geq 2^{w'}$ $(x_q \in [x_1, x_N])$. *The condition* $f_k^*(x_{i_1}) < f_k^*(x_{i_2})$ *holds if and only if* $x_{i_1} < x_{i_2}$.

Proof. We first prove that if the condition $f_k^*(x_{i_1}) < f_k^*(x_{i_2})$ holds, then $x_{i_1} < x_{i_2}$. We prove it by contradiction. If $x_{i_1} \geq x_{i_2}$, we have

$$f_k^*(x_{i_1}) = f_k^*(x_{i_2}) + \sum_{q=i_2+1}^{i_1} \frac{h_k(x_q)}{2^{w'}} \geq f_k^*(x_{i_2})$$

Second, we prove that if the condition $x_{i_1} < x_{i_2}$ holds, then $f_k^*(x_{i_1}) < f_k^*(x_{i_2})$. Similar as the proof of the property collision resistance, we have

$$f_k^*(x_{i_2}) = f_k^*(x_{i_1}) + \sum_{q=i_1+1}^{i_2} \frac{h_k(x_q)}{2^{w'}} > f_k^*(x_{i_1})$$

□

Collision Resistance: *Assume any* $h_k(x_q) \geq 2^{w'}$ $(x_q \in [x_1, x_N])$. *It is impossible to find* x_{i_1} *and* x_{i_2} *where* $x_{i_1} \neq x_{i_2}$ *such that* $f_k^*(x_{i_1}) = f_k^*(x_{i_2})$.

Proof. Without loss of generalization, we assume $i_1 < i_2$. Hence, we have

$$\begin{aligned} f_k^*(x_{i_2}) &= \frac{\sum_{q=1}^{i_1} h_k(x_q)}{2^{w'}} + \frac{\sum_{q=i_1+1}^{i_2} h_k(x_q)}{2^{w'}} \\ &= f_k^*(x_{i_1}) + \sum_{q=i_1+1}^{i_2} \frac{h_k(x_q)}{2^{w'}}. \end{aligned}$$

Because for any $h_k(x_q)$, $2^{w'} \leq h_k(x_q)$, then $\frac{h_k(x_q)}{2^{w'}} \geq 1$. Therefore, we have $f_k^*(x_{i_2}) > f_k^*(x_{i_1})$. □

C Calculation of Detection Probability

We first compute the number of choices for deleting a data item in all query results that will be detected by customers. Recall Case 1 in Section 3.4.2. Given a range query $[a, b]$, deleting a data item in B_i will be detected by the customer if $[a, b]$ is the superset of B_i, *i.e.*, $B_i \subseteq [a, b]$. For ease of presentation, let $[l_i, h_i]$ denote a bucket B_i. A bucket B_i is called a *single-value bucket* if $l_i = h_i$. For the bucket $B_i = [l_i, h_i]$, there are $l_i - x_1 + 1$ distinct values which are less than or equal to l_i. Similarly, there are $x_N - h_i + 1$ distinct values which are larger than or equal to h_i. Thus, the total number of queries, which are the supersets of $[l_i, h_i]$, can be computed as $(l_i - x_1 + 1)(x_N - h_i + 1)$. Let $e(x)$ denote the frequency of the data item with value x. The number of data items with different values satisfying a query $[a, b]$ can be computed as $\sum_{x=a}^{b} e(x)$. Thus, for deleting a data item in bucket B_i that will be detected by customers, the number of choices can be computed as

$$\pi(B_i) = (l_i - x_1 + 1)(x_N - h_i + 1) \sum_{x=l_i}^{h_i} e(x) \tag{7.3}$$

Therefore, for deleting a data item in all buckets $B_1, \cdots, B_m$ that will be detected by customers, the number of choices can be computed as

$$\pi = \sum_{i=1}^{m} \pi(B_i) = \sum_{i=1}^{m} \sum_{x=l_i}^{h_i} (l_i - x_1 + 1)(x_N - h_i + 1)e(x) \tag{7.4}$$

In our context, $e(x)$ is either equal to 0 or 1 because if multiple data items have the same value, the organization simply represents them as one data item annotated with the number of items that share this value. If there is no data item satisfying the query $[a, b]$, $\sum_{x=a}^{b} e(x) = 0$. Similarly, the number of choices for deleting a data item in all query results can be computed as

$$\pi^* = \sum_{a=x_1}^{x_N} \sum_{b=a}^{x_N} \sum_{x=a}^{b} e(x) \tag{7.5}$$

Let $I_j()$ $(1 \leq j \leq n)$ denote an indicator function as

$$I_j(x) = \begin{cases} 1 & \text{if } x = d_j \\ 0 & \text{otherwise} \end{cases}$$

We have $e(x) = \sum_{j=1}^{n} I_j(x)$. Thus, π^* can be transformed to

$$\begin{aligned} \pi^* &= \sum_{a=x_1}^{x_N} \sum_{b=a}^{x_N} \sum_{x=a}^{b} \sum_{j=1}^{n} I_j(x) = \sum_{j=1}^{n} \sum_{a=x_1}^{x_N} \sum_{b=a}^{x_N} \sum_{x=a}^{b} I_j(x) \\ &= \sum_{j=1}^{n} \sum_{a=x_1}^{d_j} \sum_{b=d_j}^{x_N} 1 = \sum_{j=1}^{n} (d_j - x_1 + 1)(x_N - d_j + 1) \end{aligned} \tag{7.6}$$

Thus, the probability that a deletion operation of the cloud provider can be detected is

$$\begin{aligned} Pr &= \frac{\pi}{\pi^*} = \frac{\sum_{i=1}^{m} \pi(B_i)}{\pi^*} \\ &= \frac{\sum_{i=1}^{m} \sum_{x=l_i}^{h_i} (l_i - x_1 + 1)(x_N - h_i + 1) e(x)}{\sum_{j=1}^{n} (d_j - x_1 + 1)(x_N - d_j + 1)} \end{aligned} \tag{7.7}$$

D Proof of Theorems 3 and 4

Proof of Theorem 3

Proof. First, we prove that if each data item d_j $(1 \leq j \leq n)$ forms a single-value bucket, then $Pr_{max} = 100\%$. Obviously, $Pr_{max} = 100\%$ is equivalent to $\pi = \pi^*$. Thus, we only need to prove $\pi = \pi^*$. For each bucket $[d_j, d_j]$ $(1 \leq j \leq n)$, because $\sum_{x=d_j}^{d_j} e(x) = 1$, we have $\pi([d_j, d_j]) = (d_j - x_1 + 1)(x_N - d_j + 1)$. For each empty bucket B_i, because $\sum_{x=l_i}^{h_i} e(x) = 0$, we have $\pi(B_i) = 0$. Thus, according to Equation 7.4, we have

$$\pi = \sum_{j=1}^{n} (d_j - x_1 + 1)(x_N - d_j + 1) = \pi^*$$

Second, we prove that if $Pr_{max} = 100\%$, each data item d_j $(1 \leq j \leq n)$ should form a single-value bucket. We prove it by contradiction. If a data item d_j does not form a single-value bucket, $Pr = \pi/\pi^* < 100\%$, which is equivalent to $\pi < \pi^*$. Assuming that $B(d_j) = [l^*, h^*]$ is not a single-value bucket, we consider the following two cases.

(1) If $B(d_j)$ includes only one data item d_j, *i.e.*, $\sum_{x=l^*}^{h^*} e(x) = 1$, the condition $l^* < d_j < h^*$ must hold. We have

$$\begin{aligned} \pi(B(d_j)) &= (l^* - x_1 + 1)(x_N - h^* + 1) \\ &< (d_j - x_1 + 1)(x_N - d_j + 1) \end{aligned}$$

Therefore, $\pi < \sum_{j=1}^{n} (d_j - x_1 + 1)(x_N - d_j + 1) = \pi^*$.

(2) If $B(d_j)$ includes $n_2 - n_1 + 1$ data items, $d_{n_1}, d_{n_1+1}, \cdots, d_j, \cdots, d_{n_2}$, $(1 < n_2 - n_1 + 1 < n)$, the condition $l^* \leq d_{n_1} < d_{n_2} \leq h^*$ must hold. We have

$$\begin{aligned} \pi(B(d_j)) &= (l^* - x_1 + 1)(x_N - h^* + 1) \sum_{x=l^*}^{h^*} e(x) \\ &< \sum_{j=n_1}^{n_2} (d_j - x_1 + 1)(x_N - d_j + 1) \end{aligned}$$

Thus, $\pi < \sum_{j=1}^{n} (d_j - x_1 + 1)(x_N - d_j + 1) = \pi^*$. □

Proof of Theorem 4

Proof. Obviously, Pr_{min} is equivalent to $\pi = n$. Thus, we only need to prove that $\pi = n$ if and only if there is only one bucket $[x_1, x_N]$.

First, we prove that if there is only one bucket $[x_1, x_N]$, then $\pi = n$.

$$\pi = \pi(B_1) = (x_1 - x_1 + 1)(x_N - x_N + 1) \sum_{x=x_1}^{x_N} f(x) = n$$

Second, we prove that if $\pi = n$, then there is only one bucket $[x_1, x_N]$. We prove it by contradiction. If there are multiple buckets $B_1, \cdots, B_m$ $(m \geq 2)$, then $\pi > n$. Let $[l_i, h_i]$ denote a bucket B_i $(1 \leq i \leq m)$. All buckets must satisfy the following condition, $x_1 = l_1 \leq h_1 < l_2 \leq h_2 < \cdots < l_m \leq h_m = x_N$. Thus, for each bucket B_i $(1 \leq i \leq m)$,

$$\pi(B_i) = (l_i - x_1 + 1)(x_N - h_i + 1) \sum_{x=l_i}^{h_i} e(x) > \sum_{x=l_i}^{h_i} e(x)$$

Thus, we have

$$\pi = \sum_{i=1}^{m} \pi(B_i) > \sum_{i=1}^{m} \sum_{x=l_i}^{h_i} e(x) = n$$

□

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Amazon web services, aws.amazon.com.

[2] Firewall throughput test, http://www.hipac.org/performance_tests/results.html.

[3] Google app engine, code.google.com/appengine.

[4] Intel lab data. http://berkeley.intel-research.net/ labdata.

[5] Microsoft azure, www.microsoft.com/azure.

[6] Rise project. http://www.cs.ucr.edu/ rise.

[7] Stargate gateway (spb400). http://www.xbow.com.

[8] Tossim. http://www.cs.berkeley.edu/ pal/research/ tossim.html.

[9] Rakesh Agrawal, Alexandre Evfimievski, and Ramakrishnan Srikant. Information sharing across private databases. In *Proc. ACM Inte. Conf. on Management of Data (SIGMOD)*, pages 86–97, 2003.

[10] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Order preserving encryption for numeric data. In *Proc. ACM Inte. Conf. on Management of Data (SIGMOD)*, pages 563–574, 2004.

[11] Ehab Al-Shaer and Hazem Hamed. Discovery of policy anomalies in distributed firewalls. In *IEEE INFOCOM'04*, pages 2605–2616, March 2004.

[12] Ehab Al-Shaer, Will Marrero, Adel El-Atawy, and Khalid ElBadawi. Network configuration in a box: Towards end-to-end verification of network reachability and security. In *Proc. IEEE Inte. Conf. on Network Protocols (ICNP)*, 2009.

[13] Sruthi Bandhakavi, Sandeep Bhatt, Cat Okita, and Prasad Rao. Analyzing end-to-end network reachability. In *Proc. IFIP/IEEE Inte. Conf. on Symposium on Integrated Network Management*, 2009.

[14] Theophilus Benson, Aditya Akella, and David Maltz. Unraveling the complexity of network management. In *Proc. USENIX Symposium on Networked Systems Design and Implementation*, 2009.

[15] Burton Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of ACM*, 13(7):422–426, 1970.

[16] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O'Neill. Order-preserving symmetric encryption. In *Proc. Inte. Conf. on Advances in Cryptology: the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 224–241, 2009.

[17] Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In *Proc. Theory of Cryptography Conference (TCC)*, pages 535–554, 2007.

[18] Justin Brickell and Vitaly Shmatikov. Privacy-preserving graph algorithms in the semi-honest model. In *Proc. Inte. Conf. on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, pages 236–252, 2010.

[19] Martin Casado, Tal Garfinkel, Aditya Akella, Michael J. Freedman, Dan Boneh, Nick McKeown, and Scott Shenker. Sane: A protection architecture for enterprise networks. In *Proc. Usenix Security Symposium*, 2006.

[20] Yeim-Kuan Chang. Fast binary and multiway prefix searches for packet forwarding. *Computer Networks*, 51(3):588–605, 2007.

[21] Fei Chen, Bezawada Bruhadeshwar, and Alex X. Liu. A cross-domain privacy-preserving protocol for cooperative firewall optimization. In *Proc. IEEE Conf. on Computer Communications (INFOCOM)*, 2011.

[22] Hong Chen, Xiaonan Man, Windsor Hsu, Ninghui Li, and Qihua Wang. Access control friendly query verification for outsourced data publishing. In *Proc. 13th European Symposium on Research in Computer Security (ESORICS)*, pages 177–191, 2008.

[23] Jerry Cheng, Hao Yang, Starsky H.Y. Wong, and Songwu Lu. Design and implementation of cross-domain cooperative firewall. In *Proc. IEEE Inte. Conf. on Network Protocols (ICNP)*, 2007.

[24] Weiwei Cheng, HweeHwa Pang, and Kian-Lee Tan. Authenticating multi-dimensional query results in data publishing. In *Data and Applications Security 2006*, pages 60–73, 2006.

[25] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press.

[26] Peter Desnoyers, Deepak Ganesan, Huan Li, and Prashant Shenoy. Presto: A predictive storage architecture for sensor networks. In *Proc. 10th Workshop on Hot Topics in Operating Systems (HotOS)*, 2005.

[27] Premkumar Devanbu, Michael Gertz, Charles Martel, and Stuart G. Stubblebine. Authentic data publication over the internet. *Journal of Computer Security*, 11(3):291–314, 2003.

[28] Qunfeng Dong, Suman Banerjee, Jia Wang, Dheeraj Agrawal, and Ashutsh Shukla. Packet classifiers in ternary CAMs can be smaller. In *Proc. ACM Sigmetrics*, pages 311–322, 2006.

[29] D. Eastlake and P. Jones. Us secure hash algorithm 1 (sha1). *RFC 3174*, 2001.

[30] E. Allen Emerson. Temporal and modal logic. 1990.

[31] Edward A. Fox, Qi Fan Chen, Amjad M. Daoud, and Lenwood S. Heath. Order-preserving minimal perfect hash functions and information retrieval. *ACM Transactions on Information Systems*, 9:281–308, 1991.

[32] A. Frank and A. Asuncion. UCI machine learning repository, 2010.

[33] Michael Freedman, Kobi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In *Proc. Inte. Conf. on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 1–19, 2004.

[34] Oded Goldreich. *Secure multi-party computations*. Working draft. Version 1.4 edition, 2002.

[35] Oded Goldreich. *Foundations of Cryptography: Volume II (Basic Applications)*. Cambridge University Press, 2004.

[36] Mohamed G. Gouda and Alex X. Liu. Firewall design: consistency, completeness and compactness. In *Proc. 24th IEEE Inte. Conf. on Distributed Computing Systems (ICDCS-04)*, pages 320–327, March 2004.

[37] Mohamed G. Gouda and Alex X. Liu. Structured firewall design. *Computer Networks Journal (Elsevier)*, 51(4):1106–1120, March 2007.

[38] Pankaj Gupta. *Algorithms for Routing Lookups and Packet Classification*. PhD thesis, Stanford University, 2000.

[39] Pankaj Gupta and Nick McKeown. Algorithms for packet classification. *IEEE Network*, 15(2):24–32, 2001.

[40] Hakan Hacigümüş, Bala Iyer, Chen Li, and Sharad Mehrotra. Executing sql over encrypted data in the database-service-provider model. In *Proc. ACM Inte. Conf. on Management of Data (SIGMOD)*, pages 216–227, 2002.

[41] Bijit Hore, Sharad Mehrotra, and Gene Tsudik. A privacy-preserving index for range queries. In *Proc. 30th Inte. Conf. on Very Large Data (VLDB)*, pages 720–731, 2004.

[42] Kyle Ingols, Richard Lippmann, and Keith Piwowarski. Practical attack graph generation for network defense. In *Proc. Annual Computer Security Applications Conf. (ACSAC)*, 2006.

[43] Zeus Kerravala. As the value of enterprise networks escalates, so does the need for configuration management. Enterprise Computing & Networking, The Yankee Group Report, January 2004.

[44] Amir R. Khakpour and Alex X. Liu. Quantifying and querying network reachability. In *Proc. Inte. Conf. on Distributed Computing Systems (ICDCS)*, 2010.

[45] Lea Kissner and Dawn Song. Privacy-preserving set operations. In *Advances in Cryptology (CRYPTO)*, pages 241–257, 2005.

[46] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. Hmac: Keyed-hashing for message authentication. *RFC 2104*, 1997.

[47] Franck Le, Sihyung Lee, Tina Wong, Hyong S. Kim, and Darrell Newcomb. Detecting network-wide and router-specific misconfigurations through data mining. 2009.

[48] Alex X. Liu and Fei Chen. Collaborative enforcement of firewall policies in virtual private networks. In *Proc. Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC)*, Toronto, Canada, August 2008.

[49] Alex X. Liu and Mohamed G. Gouda. Diverse firewall design. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 19(8), 2008.

[50] Alex X. Liu and Mohamed G. Gouda. Complete redundancy removal for packet classifiers in tcams. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, in press.

[51] Alex X. Liu, Chad R. Meiners, and Eric Torng. Tcam razor: A systematic approach towards minimizing packet classifiers in tcams. *IEEE/ACM Transactions on Networking*, to appear.

[52] Alex X. Liu, Chad R. Meiners, and Yun Zhou. All-match based complete redundancy removal for packet classifiers in TCAMs. In *Proc. 27th Annual IEEE Conf. on Computer Communications (Infocom)*, April 2008.

[53] Alex X. Liu, Eric Torng, and Chad Meiners. Firewall compressor: An algorithm for minimizing firewall policies. In *Proc. 27th Annual IEEE Conf. on Computer Communications (Infocom)*, Phoenix, Arizona, April 2008.

[54] Petr Matousek, Jaroslav Rab, Ondrej Rysavy, and Miroslav Sveda. A formal model for network-wide security analysis. In *Proc. IEEE Inte. Conf. and Workshop on the Engineering of Computer Based Systems*, 2008.

[55] Chad R. Meiners, Alex X. Liu, and Eric Torng. TCAM Razor: A systematic approach towards minimizing packet classifiers in TCAMs. In *Proc. 15th IEEE Conf. on Network Protocols (ICNP)*, pages 266–275, October 2007.

[56] Chad R. Meiners, Alex X. Liu, and Eric Torng. Bit weaving: A non-prefix approach to compressing packet classifiers in TCAMs. In *Proc. IEEE Conf. on Network Protocols (ICNP)*, pages 93–102, October 2009.

[57] Chad R. Meiners, Alex X. Liu, and Eric Torng. Topological transformation approaches to optimizing tcam-based packet processing systems. In *Proc. ACM Inte. Conf. on Measurement and Modeling of Computer Systems (SIGMETRICS)*, pages 73–84, August 2009.

[58] Ralph Merkle. Protocols for public key cryptosystems. In *Proc. IEEE Symposium on Security and Privacy*, pages 122–134, 1980.

[59] Maithili Narasimha and Gene Tsudik. Authentication of outsourced databases using signature aggregation and chaining. In *Proc. Inte. Conf. on Database Systems for Advanced Applications (DASFAA)*, 2006.

[60] Silvio Micali Oded Goldreich and Avi Wigderson. How to play any mental game. In *Proc. nineteenth anual ACM Conf. on Theory of computing*, May 1987.

[61] David Oppenheimer, Archana Ganapathi, and David A. Patterson. Why do internet services fail, and what can be done about it? In *Proc. 4th USENIX Symposium on Internet Technologies and Systems (USITS)*, March 2003.

[62] HweeHwa Pang, Arpit Jain, Krithi Ramamritham, and Kian-Lee Tan. Verifying completeness of relational query results in data publishing. In *Proc. ACM Inte. Conf. on Management of Data (SIGMOD)*, pages 407–418, 2005.

[63] HweeHwa Pang and Kian-Lee Tan. Authenticating query results in edge computing. In *Proc. 20th Inte. Conf. on Data Engineering*, pages 560–571, 2004.

[64] Stephen C. Pohlig and Martin E. Hellman. An improved algorithm for computing logarithms over gf(p) and its cryptographic significance. *IEEE Transactions Information and System Security*, IT-24:106–110, 1978.

[65] Sylvia Ratnasamy, Brad Karp, Scott Shenker, Deborah Estrin, Ramesh Govindan, Li Yin, and Fang Yu. Data-centric storage in sensornets with ght, a geographic hash table. *Mobile Networks and Applications*, 8(4):427–442, 2003.

[66] R. Rivest. The md5 message-digest algorithm. *RFC 1321*, 1992.

[67] David K. Hess David R. Safford and Douglas Lee Schales. Secure RPC authentication (SRA) for TELNET and FTP. Technical report, 1993.

[68] Yingpeng Sang and Hong Shen. Efficient and secure protocols for privacy-preserving set operations. *ACM Transactions on Infomation and System Security*, 13:9:1–9:35, 2009.

[69] Bo Sheng and Qun Li. Verifiable privacy-preserving range query in two-tiered sensor networks. In *Proc. IEEE Inte. Conf. on Computer Communications (INFOCOM)*, pages 46–50, 2008.

[70] Bo Sheng, Qun Li, and Weizhen Mao. Data storage placement in sensor networks. In *Proc. 7th ACM Inte. Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, pages 344–355, 2006.

[71] Bo Sheng, Chiu C. Tan, Qun Li, and Weizhen Mao. An approximation algorithm for data storage placement in sensor networks. In *Proc. Inte. Conf. on Wireless Algorithms, Systems and Applications (WASA)*, pages 71–78, 2007.

[72] Elaine Shi, John Bethencourt, T-H. Hubert Chan, Dawn Song, and Adrian Perrig. Multi-dimensional range query over encrypted data. In *Proc. IEEE Symposium on Security and Privacy (S&P)*, pages 350–364, 2007.

[73] Jing Shi, Rui Zhang, and Yanchao Zhang. Secure range queries in tiered sensor networks. In *Proc. IEEE Inte. Conf. on Computer Communications (INFOCOM)*, 2009.

[74] Sumeet Singh, Florin Baboescu, George Varghese, and Jia Wang. Packet classification using multidimensional cutting. In *Proc. ACM SIGCOMM*, pages 213–224, 2003.

[75] Yu-Wei Eric Sung, Carsten Lund, Mark Lyn, Sanjay Rao, and Subhabrata Sen. Modeling and understanding end-to-end class of service policies in operational networks. In *Proc. SIGCOMM*, pages 219–230, 2009.

[76] Avishai Wool. A quantitative study of firewall configuration errors. *IEEE Computer*, 37(6):62–67, 2004.

[77] Geoffery G. Xie, Jibin Khan, David A. Maltz, Hui Zhang, Albert Greenberg, Gísli Hjálmtýsson, and Jennifer Rexford. On static reachability analysis of ip networks. In *Proc. Annual Joint Conference of the IEEE Computer and Communication Societies (INFOCOM)*, 2005.

[78] Zhiqiang Yang, Sheng Zhong, and Rebecca N. Wright. Privacy-preserving classification of customer data without loss of accuracy. In *Proc. Inte. Conf. on Data Mining (SIAM)*, 2005.

[79] Andrew C. Yao. Protocols for secure computations. In *Proc. 23rd IEEE Symposium on the Foundations of Computer Science (FOCS)*, pages 160–164, 1982.

[80] Andrew C. Yao. How to generate and exchange secrets. In *Proc. 27th IEEE Symposium on Fundations of Computer Science*, 1986.

[81] Lihua Yuan, Hao Chen, Jianning Mai, Chen-Nee Chuah, Zhendong Su, and Prasant Mohapatra. Fireman: a toolkit for firewall modeling and analysis. In *IEEE Symposium on Security and Privacy*, May 2006.

[82] Demetrios Zeinalipour-yazti, Song Lin, Vana Kalogeraki, Dimitrios Gunopulos, and Walid A. Najjar. Microhash: An efficient index structure for flash-based sensor devices. In *Proc. 4th USENIX Conf. on File and Storage Technologies (FAST)*, pages 31–44, 2005.

[83] Bo Zhang, T. S. Eugene Ng, and Guohui Wang. Reachability monitoring and verification in enterprise networks. In *Proc. SIGCOMM*, 2008.

[84] Rui Zhang, Jing Shi, and Yanchao Zhang. Secure multidimensional range queries in sensor networks. In *Proc. ACM Inte. Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, 2009.