

2010

## Secure and efficient data extraction for ubiquitous computing applications

Chiu Chiang Tan  
*College of William & Mary - Arts & Sciences*

Follow this and additional works at: <https://scholarworks.wm.edu/etd>



Part of the [Computer Sciences Commons](#)

---

### Recommended Citation

Tan, Chiu Chiang, "Secure and efficient data extraction for ubiquitous computing applications" (2010). *Dissertations, Theses, and Masters Projects*. Paper 1539623571.  
<https://dx.doi.org/doi:10.21220/s2-0nmg-xv96>

This Dissertation is brought to you for free and open access by the Theses, Dissertations, & Master Projects at W&M ScholarWorks. It has been accepted for inclusion in Dissertations, Theses, and Masters Projects by an authorized administrator of W&M ScholarWorks. For more information, please contact [scholarworks@wm.edu](mailto:scholarworks@wm.edu).

SECURE AND EFFICIENT DATA EXTRACTION FOR UBIQUITOUS  
COMPUTING APPLICATIONS

Chiu Chiang Tan

Singapore

Bachelor of Science, University of Texas at Austin, 2004  
Bachelor of Arts, University of Texas at Austin, 2004

A Dissertation presented to the Graduate Faculty  
of the College of William and Mary in Candidacy for the Degree of  
Doctor of Philosophy

Department of Computer Science

The College of William and Mary  
August, 2010

APPROVAL PAGE

This Dissertation is submitted in partial fulfillment of  
the requirements for the degree of

Doctor of Philosophy



---

Chiu Chiang Tan

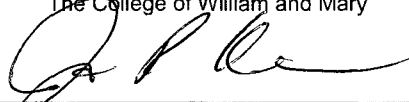
Approved by the Committee, July 2010



---

Committee Chair

Associate Professor Qun Li, Computer Science  
The College of William and Mary



---

Associate Professor Phil Kearns, Computer Science  
The College of William and Mary



---

Associate Professor Weizhen Mao, Computer Science  
The College of William and Mary



---

Associate Professor Haining Wang, Computer Science  
The College of William and Mary



---

Assistant Professor Gexin Yu, Mathematics  
The College of William and Mary

## ABSTRACT PAGE

Ubiquitous computing creates a world where computers have blended seamlessly into our physical environment. In this world, a "computer" is no longer a monitor-and-keyboard setup, but everyday objects such as our clothing and furniture. Unlike current computer systems, most ubiquitous computing systems are built using small, embedded devices with limited computational, storage and communication abilities. A common requirement for many ubiquitous computing applications is to utilize the data from these small devices to perform more complex tasks. For critical applications such as healthcare or medical related applications, there is a need to ensure that only authorized users have timely access to the data found in the small device. In this dissertation, we study the problem of how to securely and efficiently extract data from small devices.

Our research considers two categories of small devices that are commonly used in ubiquitous computing, battery powered sensors and battery free RFID tags. Sensors are more powerful devices equipped with storage and sensing capabilities that are limited by battery power, whereas tags are less powerful devices with limited functionalities, but have the advantage of being operable without battery power. We also consider two types of data access patterns, local and remote access. In local data access, the application will query the tag or the sensor directly for the data, while in remote access, the data is already aggregated at a remote location and the application will query the remote location for the necessary information. The difference between local and remote access is that in local access, the tag or sensor only needs to authenticate the application before releasing the data, but in remote access, the small device may have to perform additional processing to ensure that the data remains secure after being collected. In this dissertation, we present secure and efficient local data access solutions for a single RFID tag, multiple RFID tags, and a single sensor, and remote data access solutions for both RFID tag and sensor.

# Table of Contents

<b>Acknowledgments</b>	<b>x</b>
<b>List of Tables</b>	<b>xi</b>
<b>List of Figures</b>	<b>xii</b>
<b>1 Introduction</b>	<b>2</b>
1.1 Abstractions . . . . .	5
1.2 Problems and Contributions . . . . .	7
1.3 Organization . . . . .	12
<b>2 Secure and Serverless RFID Authentication and Search Protocols</b>	<b>13</b>
2.1 Related Work . . . . .	16
2.2 RFID Authentication . . . . .	19
2.2.1 Setup . . . . .	21
2.2.2 Authentication Protocol . . . . .	23
2.3 Security Analysis . . . . .	24
2.4 RFID Search . . . . .	27
2.4.1 Security Analysis . . . . .	30

2.4.2	Search Protocol Improvements . . . . .	31
2.5	Additional Discussion . . . . .	33
2.6	Conclusion . . . . .	35
<b>3</b>	<b>Monitoring for Missing RFID Tags</b>	<b>37</b>
3.1	Related Work . . . . .	39
3.2	Problem Formulation . . . . .	40
3.3	TRP: Trusted Reader Protocol . . . . .	42
3.3.1	Intuition and assumptions . . . . .	43
3.3.2	TRP algorithm . . . . .	44
3.3.3	Analysis . . . . .	45
3.4	UTRP: UnTrusted Reader Protocol . . . . .	47
3.4.1	Vulnerabilities . . . . .	47
3.4.2	Intuition and assumptions . . . . .	49
3.4.3	UTRP algorithms . . . . .	51
3.4.4	Analysis . . . . .	52
3.5	Evaluation . . . . .	56
3.6	Conclusion . . . . .	58
<b>4</b>	<b>Microsearch : A Search Engines for Small Devices</b>	<b>61</b>
4.1	Related Work . . . . .	65
4.2	System Architecture . . . . .	68
4.2.1	Microsearch Design . . . . .	68
4.3	Secure Microsearch . . . . .	72

4.3.1	Threats . . . . .	72
4.3.2	Straw Man Protocols . . . . .	73
4.3.3	Single User Protocol . . . . .	76
4.3.4	Multiple users . . . . .	78
4.3.5	Performance Details . . . . .	78
4.4	Query Resolution . . . . .	79
4.4.1	Information Retrieval Basics . . . . .	80
4.4.2	Basic Algorithm . . . . .	81
4.4.3	Performance Improvements . . . . .	82
4.4.4	Space Efficient Algorithm . . . . .	83
4.5	Theoretical Model . . . . .	85
4.6	System Evaluation . . . . .	89
4.6.1	Hardware and Implementation . . . . .	89
4.6.2	Generating Workload Data . . . . .	90
4.6.3	System Performance . . . . .	91
4.6.4	Search Accuracy . . . . .	93
4.6.5	Experiment Limitations . . . . .	96
4.6.6	Model Accuracy . . . . .	97
4.6.7	Alternative Design . . . . .	97
4.7	Conclusion . . . . .	101
<b>5</b>	<b>Privacy Protection for RFID-based Tracking Systems</b>	<b>103</b>
5.1	Related Work . . . . .	105
5.2	Problem Formulation . . . . .	107

5.2.1	Adversary model . . . . .	108
5.3	Strawman solutions . . . . .	109
5.3.1	Strawman 1 . . . . .	109
5.3.2	Strawman 2 . . . . .	110
5.3.3	Discussion . . . . .	112
5.4	RFID Protocol . . . . .	114
5.4.1	Collecting data from tag . . . . .	115
5.4.2	Querying the database . . . . .	117
5.4.3	Security analysis . . . . .	118
5.4.4	Protocol discussion . . . . .	122
5.5	Additional Discussion . . . . .	122
5.5.1	Detect deleted data . . . . .	123
5.5.2	Detect tampered data . . . . .	124
5.6	Conclusion . . . . .	126
<b>6</b>	<b>IBE-Lite: A Lightweight Identity Based Cryptography for Body Sensor Networks</b>	<b>127</b>
6.1	Related Work . . . . .	131
6.2	IBE-Lite Solution . . . . .	132
6.2.1	IBE-Lite . . . . .	133
6.2.2	BSN Security Protocols . . . . .	134
6.2.3	Query Improvements . . . . .	137
6.3	Analysis and Evaluation . . . . .	138
6.3.1	Security . . . . .	138
6.3.2	Performance . . . . .	141



6.4	Conclusion . . . . .	144
<b>7</b>	<b>Conclusion and Future Work</b>	<b>146</b>
7.1	Future work . . . . .	147
	<b>Bibliography</b>	<b>150</b>
	<b>Vita</b>	<b>158</b>

To my family.

## ACKNOWLEDGMENTS

This dissertation would not have been possible without my advisor, Dr. Qun Li. His quick mind and critical insights have sharpened my ability to select good research problems, and his enthusiasm and encouragement have helped me work through the difficulties I encountered in my research. I am grateful for the freedom he gave me to pursue my own research interests and directions even when they diverged from his own. I hope I can give my future students the same level of freedom and support as Qun gave me.

I would like to thank the members of my dissertation committee, Dr. Phil Kearns, Dr. Weizhen Mao, Dr. Haining Wang, and Dr. Gexin Yu, for their time and effort in guiding me through the various stages leading up to this dissertation. Their comments and suggestions have greatly improved this dissertation. I would like to give special thanks to Dr. Virginia Torczon, Dr. Weizhen Mao, Dr. Haining Wang, and Dr. Kui Ren, for their assistance during my job application process.

I have been fortunate to work with many brilliant people during my stay at William and Mary. I would like to thank Dr. Bo Sheng for the many enlightening discussions we have had, and I would also like to extend my thanks to his wife, Dr. Ningfang Mi, for putting up with the long hours Bo and I spent working together. I would like to thank Dr. Haodong Wang for all his patience and help during our collaborations. It has been my pleasure to collaborate with Dr. Lei Xie during his short stay at William and Mary. I would like to thank the remaining members of my research group, Fengyuan Xu, Hao Han, Yifan Zhang, Wei Wei, and Zhengrui Qin, for their support and friendship. I would not have enjoyed my studies as much without the friendship of Dr. Mengjun Xie, Chuan Yue, and Zhenyu Wu.

I am grateful to all the past and present staff in the Computer Science Department. In particular, I would like to thank Vanessa Godwin and Jacquelyn Johnson for all they have done to make my stay at the Department as stress-free as possible. I would also like to thank the Techies for all their efforts in keeping everything running smoothly.

Finally, I would like to thank my family for all they have done over the years. While they may not understand what I have been doing all these years, they have supported me nonetheless. Their unwavering support has been, and continues to be, a source of strength and comfort.

# List of Tables

2.1	Notations . . . . .	22
3.1	Notations . . . . .	42
4.1	Parameters for RSA and ECC encryption. . . . .	79
4.2	System Model Variables . . . . .	86
4.3	Recommended size of main index ( $H$ ) for different query response requirements . . . . .	98
5.1	Unencrypted table in database . . . . .	107
5.2	Database table from Strawman 1 . . . . .	110
5.3	Database table by Strawman 2 . . . . .	111
5.4	Summary of Notations . . . . .	114
5.5	Table maintained by $TS$ . . . . .	116
5.6	Table maintained by $LS$ . . . . .	117
5.7	Table maintained by user . . . . .	118
6.1	Size of basic ECC primitives. . . . .	132

# List of Figures

1.1	Summary of research work . . . . .	7
3.1	Vulnerability of TRP . . . . .	48
3.2	Re-seeding after first reply . . . . .	50
3.3	Re-seeding just from slot 2 . . . . .	50
3.4	Comparing <i>collect all</i> versus TRP . . . . .	57
3.5	Accuracy of TRP with $\alpha = 0.95$ . . . . .	58
3.6	Comparing TRP versus UTRP . . . . .	59
3.7	Accuracy of UTRP with $\alpha = 0.95$ . . . . .	60
4.1	(a) Utilizing backend server. (b) No backend server . . . . .	63
4.2	(a) 1) Flushes tuples from buffer cache, 2) Copies address , $addr_{17}$ , into inverted index. (b) 1) Copies previous metadata page address from inverted index. 2) Flushes tuples from buffer cache. 3) Copies new address, $addr_{26}$ , into inverted index. . . . .	70
4.3	Value of $x$ and $E'$ v.s. Index Size ( $H$ ): Buffer cache is $5K$ bytes, buffer size $B$ is 640. The flat line illustrates the value of $E = 31$ . . . . .	88

4.4	Query Performance v.s. Index Size ( $H$ ): $D = 1000, m = 10, t = 1, E = 31, B = 640$ (5K-byte buffer). . . . .	88
4.5	Insert Performance v.s. Index Size ( $H$ ): We set $D = 1000, m = 10, E = 31, B = 640$ (5K-byte buffer). . . . .	89
4.6	Term distribution for annotation workload . . . . .	92
4.7	Predicted and actual insert performance . . . . .	93
4.8	Predicted and actual query response time measured in seconds . . . . .	94
4.9	Processing time overhead of search system processing . . . . .	95
4.10	Query accuracy ( $k=3$ ) . . . . .	96
4.11	Actual query response time. . . . .	98
4.12	Comparing alternative scheme with our scheme . . . . .	99
4.13	Comparing power consumption of our scheme verses alternative scheme . . . . .	100
5.1	Typical RFID tracking system . . . . .	104
5.2	Strawman protocol 1. The tag ID is 101. . . . .	109
5.3	Strawman protocol 2. The tag ID is 101. . . . .	111
5.4	Obtaining data from tags, and querying databases for data. . . . .	115
5.5	Reader-tag interaction. The dotted line in step (3) denotes that the reader transmits directly to the timestamp server, bypassing the location server. . . . .	115
6.1	Amount of storage needed to store $n$ keys for different encryption methods. . . . .	142
6.2	Data transmission overhead for different encryption schemes. All values in bytes. . . . .	143
6.3	Time needed to derive one $y_{str}$ using different $n$ number of public keys, $y_1, \dots, y_n$ . . . . .	144

6.4 Comparing the difference in number of message passed during search when using  
hints verses no hints. . . . . 145

SECURE AND EFFICIENT DATA EXTRACTION FOR UBIQUITOUS  
COMPUTING APPLICATIONS



# Chapter 1

## Introduction

Ubiquitous computing is the paradigm that considers an environment where people are surrounded by numerous computing devices. In an ubiquitous computing environment, a “computer” is no longer a laptop or a smartphone, but everyday items such as our clothing and furniture. In the words of Mark Weiser, the father of ubiquitous computing, computers in ubiquitous computing will “*weave themselves into the fabric of everyday life until they are indistinguishable from it*” .

This new type of computing entails embedding small devices into everyday physical objects. These small devices are essentially miniature computing devices, capable of storing information about the attached physical object, processing the information, and communicating that information with a user via a wireless interface. For instance, a small device attached to a carton of milk can store not only the expiry date of the milk, but also maintain a record of the temperature in which the milk was kept in. A user can access the information contained within this small device to determine whether the carton of milk has gone bad. For more critical ubiquitous computing applications, such as those used in healthcare or medical scenarios, the requirements for extracting data from these small devices are much higher. We illustrate some of these applications using the following example scenarios.

- Alice is staying at home to recover from an illness. She is outfitted with clothes that have been embedded with small devices that will continuously collect physiological data such as blood pressure and heart rate. The collected information will be periodically relayed to her doctors via the Internet to allow them to monitor her condition remotely. This system is convenient, since it frees Alice from having to make frequent trips to the hospital, and also provides higher quality of care, because the doctors can have timely access to information regarding her condition.
- Alice's condition takes a turn for the worst and she faints in her kitchen. She is unconscious and unable to call for help. The small devices in her clothes detect the sudden acceleration, while the small devices embedded in her kitchen floor detects the lack of movement. A warning message is broadcast from Alice's clothing to her home server, which in turn sends for help. Without such an ubiquitous system in place, Alice would most likely have to remain unconscious for a longer period of time until being discovered by a neighbor or a friend.
- Upon admission to the hospital, Alice is outfitted with a bracelet that contains a unique ID that is associated with her stay in the hospital. Medical treatments and documents associated with Alice, such as her pharmaceuticals and blood transfusion bags, are similarly affixed with a small device containing the same unique ID. Doctors and nurses will verify that the ID on the small devices matches Alice's bracelet before administering any treatment. These small devices complement traditional hospital charts and records to act as a check against human errors.

From these examples, it is clear that while these ubiquitous computing applications are ben-

eficial, they also create new types of risks and dangers. An application that allows a doctor to effortlessly monitor someone remotely can just as easily allow malicious parties the same capabilities without adequate security. One common requirement among many of these applications is the need to extract data from these small devices. In this dissertation, we focus on the problem of *securely* and *efficiently* extracting information from these small devices.

We define security as ensuring that only authorized users are allowed access to the information contained within the small device. We define two types of efficiency in this dissertation, the speed of retrieving the relevant information from a large amount of data, and the energy expended by the small device when extracting the information. The motivation for studying the former is that while the small device may contain a large amount of information, only a smaller portion of the data may be relevant at a given time. Data extraction solutions that extract unnecessary data will result in poor overall performance. The motivation for the latter consideration is that many small devices have limited power supply, and extraction techniques that are power inefficient will quickly render the small devices inoperable.

The concepts of security and efficiency are closely related with each other. A system or algorithm design that considers only one and not the other is unlikely to work well in practice. For example, consider a security scheme that programs the small devices to encrypt all their collected data with different encryption keys. Such a scheme may be secure, but since the ubiquitous computing application is comprised of thousands of small devices, such a design will require the application to manage a large number of keys, which is clearly inefficient. We need to consider both security and efficiency to arrive at better system and algorithm designs.

## 1.1 Abstractions

To develop secure and efficient data extraction solutions suitable for different ubiquitous computing applications, we need to create abstractions to represent the key characteristics of these applications. We make two abstractions based on our observations. The first abstraction is to categorize the hardware of a *small device*. The second abstract is to characterize how these small devices are accessed.

In this dissertation, we classify a “small device” into two categories based on whether or not the small device is battery powered. We term the non-battery powered small device as an *RFID tag*, and the battery powered small device as a *sensor*. In general, an ubiquitous computing application will attach an RFID tag to an object when the application needs to simply identify that object. A sensor will be attached to the object when the application needs to either sense some data about a physical object such as temperature, or when the application needs to store large amounts of data onto a physical object.

An RFID tag is a battery free device that powers itself through the RFID reader’s broadcast signal. This is done either through inductive coupling for high frequency (HF) RFID tags , or electromagnetic backscatter for ultra high frequency (UHF) tags. The main difference between the two lies in the communication range, with the HF tag having a range of several centimeters, while the UHF tag have a range of approximately a dozen meters. This method of supplying energy limits the computational abilities of the RFID tag to very basic functions, such as bit string comparisons and reading an ID number for persistent storage. Some of the more advanced RFID tags can perform additional functions such as writing to persistent storage, perform hashing and encryption. In general, the security challenge for a tag is related to the limited computational ability of the tag to perform conventional cryptographic operations. The efficiency challenge lies

in the large numbers of tags that have to be processed. Given the low cost of an RFID tag compared to a sensor, we can generally assume that there will be more tags than sensors.

A sensor is a battery powered device than can perform more complex operations than a tag. A basic sensor is equipped with a microcontroller, memory, persistent storage, and a wireless radio. The communication range of a sensor is typically under 100 meters. In addition, a sensor can be equipped with specialized sensing capabilities to measure variables such as blood glucose, temperature, and acceleration. The sensor depends on its internal battery to power all operations. A sensor with a depleted battery can only maintain the data stored in the persistent storage of the sensor. Since the sensor has a general purpose microcontroller, it is capable of performing conventional cryptographic operations such as verifying a digital certificate. The general security challenge when dealing with a sensor lies in balancing the security operations with the limited battery capacity of the sensor. The general efficiency challenge lies in managing the large amounts of information stored in a sensor using only the limited amounts of memory and battery power available.

There are two general ways an ubiquitous computing application can access the data contained within a small device. The application can either access the data directly from the small device to perform addition operations, or the application can access the data *after* the data from the small device has been aggregated to a central depository. We term the former access as *local* access, and the latter as *remote* access.

From the standpoint of a small device, there is a crucial difference between local and remote access. In local access, the small device needs only to authenticate the application before releasing the information. In remote access, the small device has to ensure that the information aggregated at the depository remains secure and can be access efficiently. This may require the small device

to do additional processing before releasing the information.

There are two types of local access for RFID tags. The first is when we want to extract data from just one RFID tag. This represents applications such as reading an RFID tag affixed to a passport or driver's license. The second is when we want to extract data from many RFID tags. This represents applications such as inventory control where we need to extract information from thousands of RFID tags. We consider the scenario for remote access of RFID tags as when the all the tag data has already been aggregated to an RFID tag database. The local access for sensors considers the scenario where descriptive information are stored directly into the sensor, and the user is expected to be able to query for specific information from the sensor. Remote access for a sensor considers applications where the sensor data is aggregated to a backend server for future retrieval by an ubiquitous computing application.

## 1.2 Problems and Contributions

This dissertation presents five problems corresponding to the abstractions given earlier. A summary of these problems is shown in Fig. 1.1. The details about the problem are as follows.

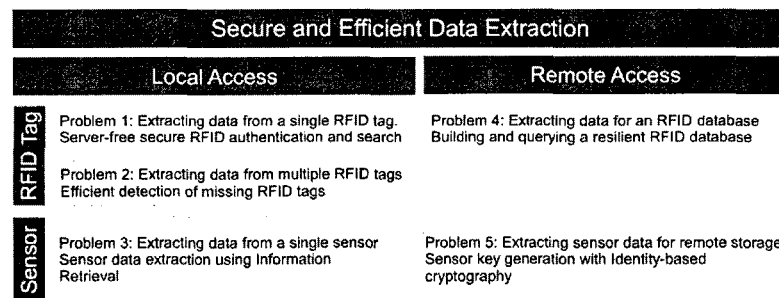


Figure 1.1: Summary of research work

1. **Local RFID data access with single tag** Conventional RFID security protocols usually

have three components, a tag, a reader, and a backend server. The backend server contains the secrets for all the RFID tags. The reader will first query the RFID tag to obtain a ciphertext, which the reader forwards to the backend server. The server will decrypt the ciphertext to obtain the RFID tag's ID, and return the ID after authenticating the reader. This prevents unauthorized readers from accessing the tag's ID even though the tag is unable to authenticate the reader directly.

We propose a search and authentication protocol that allows an authorized RFID reader to extract the ID from a single RFID tag. Our protocol is a server-free protocol, since the reader is able to determine the ID without contacting the backend server. Our protocol is among the earliest server-free security protocols to be proposed.

- 2. Local RFID data access with multiple tags** A common application that involve multiple RFID tags is inventory control. Here, each physical object is affixed with its own RFID tag, and each tag contains a unique ID. The tag becomes a representation of that object. The existing technique to determine whether any tags are missing is to first collect all the tag IDs and then verify them against some record. This approach, however, is unlikely to scale well as the number of RFID tags increases.

We proposed a missing tag event detection protocol that, given a large collection of RFID tags, is able to determine whether there are any missing tags. The protocol has two versions, with the first version being more efficient but requires the RFID reader to be controlled by an honest user, while the second version can return the correct answer even if the user of the RFID reader is dishonest.

- 3. Local sensor data access** A sensor will often have to store large amounts of data locally

due to the large amounts of power needed to transmit all the information. A technique to conserve power is for the sensor to efficiently index the stored data and allow users to query for the data. This way, the sensor can limit its transmission to only relevant data that users need to conserve power. Earlier search systems limited the indexing to numeric data with pre-defined upper and lower bounds. This type of search system is unsuitable for ubiquitous computing applications that require storing textual descriptions into the sensor.

We proposed a search system for a sensor that can index textual information (e.g. short notes, voice memos, photographs), and uses information retrieval techniques to resolve user queries. Our solution is the first search system for sensors that is able to index and resolve queries over textual rather than numeric data.

4. **Remote RFID data access** An RFID-based tracking system maintains a network of readers installed at different physical locations, where each reader will periodically read all the nearby RFID tags and report the collected IDs back to an RFID database. Authenticated users can query the database to determine the movements of a particular RFID tag. Existing tracking systems usually adopt the “trusted server” assumption where this database is assumed to be invulnerable to adversary compromise in order to provide privacy protections. However, this assumption is unrealistic in a real world environment.

We proposed a tracking system where a tag’s ID is stored in a database in a manner that is resilient to partial adversary compromise. Our system design allows a user with knowledge of a tag’s secret to quickly determine the whereabouts of a user, while at the same time, prevents an adversary with partial control of the database from doing the same.

5. **Remote sensor data access** A sensor, like those used in medical applications to sense phys-



iological data, will continuously collect large amounts of information, which will then be stored in a remote server for future access by various people. Due to the sensitive nature of the data, a user will not want to trust the remote server to protect his data. One option is for the sensor to encrypt each piece of data with a different key and let the user control who should have the keys to decrypt the data. However, this scheme requires complicated key management protocols to manage all the keys.

We proposed a security scheme that simplifies key management for a sensor that needs to encrypted data using many different cryptographic keys. Our scheme provides properties similar to identity-based cryptography to allow the sensor to generate keys using a textual description.

Our main contributions for each problem are as follows.

- For the local RFID access problem involving a single RFID tag, we proposed a server-free scheme where the reader will first authenticate with the backend server and obtain its own secret. Once this is completed, our protocol allows the reader to query and obtain the IDs without any further contact with the backend server. In our scheme, the tag “authenticates” the reader by generating a response that corresponds to the reader’s secret which only an authorized reader can obtain from the backend server. We also provide an efficient search protocol where an authorized reader that wants to find a particular tag can search directly for that tag without having to query all the other RFID tags.
- For the local RFID access with multiple tags, we proposed a protocol that leverages the slotted ALOHA communication protocol that is part of existing RFID standards. When a reader wants to communicate with multiple RFID tags, the reader will first broadcast the

total number of frameslots available, and each RFID tag will internally compute a particular frameslot to respond in. Later, the reader will coordinate the countdown of each frameslot. A tag will keep silent until the countdown reaches its frameslot before broadcasting its ID. We let each tag to compute the frameslot as a function of its ID, and then by observing the order of the tag responses, we are able to determine whether there are any missing tags. Our scheme is more efficient since it is not affected by multiple tags picking the same frameslot to respond (a collision), whereas in a conventional approach, the reader will have to repeat the ID collection process in the event of a collision.

- For the local sensor access problem, we designed a system that allows the sensor to efficiently index textual descriptions. We do not place restrictions on the vocabulary used in the description. At the heart of our system is a memory efficient query resolution algorithm that provides an upper bound on the amount of RAM needed to process the  $k$ -most relevant answers, regardless of the contents that have been indexed. This property is crucial to avoid expending energy swapping of data to and from NAND flash memory due to insufficient memory during query processing. Our solution also provides security protections against denial-of-service and spoofing attacks.
- For the remote RFID data access problem, we propose a protocol that lets an RFID tag to compute a tuple in response to each reader's query. The tuple is constructed in such a way that the two portions of the tuple are linked by a tag secret. The reader will forward the first portion of the tuple to one database, and the second portion of the tuple to another database. A legitimate user with knowledge of the tag's secret can efficiently query both databases to derive the location of the tag, but an adversary that does not know the tag secret cannot do so, even if the adversary has control over one of the databases.

- For the remote sensor data access problem, we proposed a lightweight solution with the properties similar to that of identity based cryptography but with a lower overhead. Identity based cryptography is a technique that allows the creation of public and private keys using human readable strings. Our solution uses a set of intuitive rules, such as the using the current date and time, to create a human readable string to compute the encryption keys. This allows the sensor to generate the necessary keys to encrypt the data before transmission. A user can simply re-apply the same rule later when computing the decryption keys. The use of human readable rules greatly simplifies cryptographic key management for both the sensor and the user.

### 1.3 Organization

The rest of this dissertation is organized as follows. In Chapter 2, we present our solutions to allow an RFID reader to securely access the data from an individual RFID tag. In Chapter 3, we present techniques to efficiently monitor a large set of RFID tags to determine the presence of missing tags. Chapter 4 presents our solution to efficiently extract data from a small sensor using information retrieval algorithms. In Chapter 5, we propose an efficient and privacy-preserving technique to access an RFID database used in tracking, and in Chapter 6 we proposed protocols based on lightweight identity-based cryptography to secure data collected in a sensor. Finally, we conclude in Chapter 7.

## **Chapter 2**

# **Secure and Serverless RFID**

## **Authentication and Search Protocols**

In this opening chapter, we consider local data extraction from a single RFID tag. Secure authentication and search are the basic building blocks for applications that need to securely obtain information from an RFID tag. This chapter shows how to design powerful search and authentication protocols using only very simple operations suitable for RFID tags.

Radio Frequency Identification (RFID) technology is increasingly being deployed in diverse applications ranging from inventory management to anti-counterfeiting protection [102]. Features such as the ability for a reader to read data off an RFID tag located several meters away, make RFID tags an attractive replacement for barcodes which require close proximity to a reader before being read. Nonetheless, RFID tags have yet to supplant the ubiquitous barcode found on almost every grocery product. This slow adoption is partly due to the security and privacy concerns over the pervasive deployment of RFID tags. Such concerns include the illicit tracking of RFID tags which violate the privacy of the holders of the tags. Until these concerns are adequately addressed,

large scale adoption of RFID is unlikely to materialize.

Recent work [34, 67, 82, 106] attempts to solve the RFID security and privacy problem by utilizing the “central database model”. There are three players in this model: an RFID reader, an RFID tag, and a secure central database. To obtain data from a tag, the reader first queries the tag and then forwards the tag reply to the central database. The reader obtains no useful information from the tag reply. After the database authenticates the reader and verifies that the tag reply is genuine, the database returns the tag information to the reader. While the central database approach provides security and privacy protections, it is dependent on a reliable connection between an RFID reader and the central database. Consider for example, a truck driver dispatched to an off-site location to collect some merchandise tagged with RFID tags. He has with him a PDA which doubles as an RFID reader. Due to the remote location, the truck driver is unable to connect to the central database to authenticate the goods. As a result, despite having an authorized reader and genuine RFID tags, the driver is unable to obtain the data.

A simple alternative, analogous to using a central database, is to download the information from the database onto the reader. The RFID reader can then continue to access the RFID tags as before. However, having multiple readers increases the likelihood of a stolen or misplaced reader. These compromised readers are more valuable than the readers under the central database model, since they contain information originally found only in the database. This information can include the unique ID and secret password of an RFID tag. An adversary can use this information to create fake RFID tags that are indistinguishable from the real ones. The adversary first obtains a “blank” RFID tag and then proceeds to store data from the compromised reader onto this blank tag. Since this fake tag has the same information as a real RFID tag, a reader is unable to distinguish between the two. In this chapter, we suggest protocols that provide similar security and privacy protections

as the central database model without requiring a persistent connection to the database. The protocols also prevent an adversary from using a compromised reader to create indistinguishable fake RFID tags.

After providing security and privacy protection to a single reader querying a single tag, a natural extension is to provide the same protection to situations where there is a single reader and multiple tags. One such situation is when a reader needs to search for a particular RFID tag out of a large collection of tags. As the number of RFID tags in circulation increases, the ability to search for RFID tags is invaluable when the reader only requires data from a few tags rather than all the tags in a collection. Authenticating each tag one at a time until the desired tag is found is a time consuming process. Surprisingly, the problem of RFID search has not been widely addressed in the literature, despite the availability of search capabilities in commercial RFID products. In this chapter, we examine the challenges of extending security and privacy protection to RFID search, and suggest several solutions.

We make the following three contributions in this chapter. First, we propose an authentication protocol that provides mutual authentication between the RFID reader and RFID tag without the need for a persistent central database. This is a departure from recent work on RFID security and privacy research. Second, our schemes consider security for both the RFID reader and the RFID tag. This differs from some of the earlier research which focused on only protecting the reader or the tag. Third, we introduce the problem of searching for RFID tags with security and privacy protection, and suggest several solutions.

## 2.1 Related Work

RFID security and privacy research can be broadly divided into two categories. The first category is protocol based. Its emphasis is on designing better protocols using mostly lightweight primitives [108] known to be implementable on RFID tags. Our work falls under this category. The second category is hardware based. The emphasis is on improving RFID tag hardware to provide additional security primitives like elliptic curve cryptography. For the remainder of this section, the focus is on prior work done in the first category. A brief discussion of RFID hardware improvements is given at the end. Interested readers can refer to an online resource by Avoine [6] for up-to-date information, and recent survey papers [55, 92] for more details.

Early work by Weis et al. [117] used a backend database to perform RFID authentication. A reader querying the RFID tag will receive a *metaID*. The reader forwards this *metaID* to the backend server which then retrieves the real tag ID for the reader. Every tag has a unique *metaID* and will always reply with the same *metaID* value when queried. This creates a privacy problem since an adversary can track the movements of a tag by repeatedly querying and comparing *metaID* values. The authors proposed the randomized hash lock scheme to solve this problem. Under this scheme, the tag returns  $(r, ID \oplus f_k(r))$  when queried by a reader, where  $r$  is a random number generated by the tag,  $k$  is the tag's secret key and  $f_k$  is a pseudorandom function. The reader forwards this reply to a secure database which then searches its database for the ID/secret key pair that matches the tag reply. Once found, the tag ID is returned to the reader. Since every new reader query results in a different reply, the adversary is unable to track the tag.

Molnar and Wagner [78] pointed out that the randomized hash lock scheme does not defend against an eavesdropper. An adversary can eavesdrop on the communication between reader and tag to learn the tag reply,  $(r, ID \oplus f_k(r))$ . The adversary then uses this information to impersonate

the RFID tag to fool a reader. In their paper, the authors suggest having both the reader and tag each contribute a random number,  $r_1$  and  $r_2$  respectively. Their approach assumes that the reader knows the tag secret  $k$ . After the reader and tag exchange random numbers, the tag replies with  $ID \oplus f_k(0, r_1, r_2)$ . Since the reader knows  $k$ , he can derive  $f_k(0, r_1, r_2)$  and obtain  $ID$ . The protocol works without a central database. However, it does not consider the case of a compromised reader. An adversary with a compromised reader will know the tag secret of every tag the reader has access to. The adversary can then use this information to make duplicate tags to fool other readers. Our protocols address this particular vulnerability.

Dimitriou [34] is a more recent example of a protocol based on a database. In this protocol, both the reader and tag exchange random numbers,  $n_r$  and  $n_t$ , at the start of the query. The tag then returns  $(h(ID_i), n_t, h_{ID_i}(n_t, n_r))$  to the reader, where  $ID_i$  is the tag secret. The reader learns nothing from this reply, and forwards it to the database. The database uses  $h(ID_i)$  to determine the matching tag secret  $ID_i$ . This  $ID_i$  is applied to  $n_t$  and  $n_r$  to verify the tag reply. Once satisfied, the database updates the tag secret from  $ID_i$  to  $ID_{i+1}$ . The tag information, together with  $h_{ID_{i+1}}(n_t, n_r)$ , is returned to the reader. The reader completes the protocol by forwarding  $h_{ID_{i+1}}(n_t, n_r)$  back to the tag. The tag determines  $ID_{i+1}$  independently, and applies it to the two random numbers used earlier. If the result matches  $h_{ID_{i+1}}(n_t, n_r)$ , the tag knows that the reader has been authenticated by the database. The tag updates its secret to  $ID_{i+1}$  and the protocol terminates. Otherwise, the tag retains the old secret  $ID_i$ . Similar protocols [67, 82] also use the idea of changing the tag secret after every query. A key feature of this protocol is how desynchronization between tag and server is avoided. A fake RFID tag will not be able to generate a reply to convince the database to update the tag secret  $ID_i$ . A rogue reader is unable to derive  $h_{ID_{i+1}}(n_t, n_r)$  to convince an RFID tag to change its secret. Work by [69, 70] examines desynchronization attacks in greater detail.



While RFID with database protocols are relatively new, a similar problem is found in 3GPP mobile authentication [50, 123]. In 3GPP authentication, mutual authentication is required between the mobile user and network. Synchronization of sequence numbers used by a mobile user and the home network is also required. These requirements are similar to the mutual authentication between a reader and a tag, and the synchronization of tag secret between the database and the RFID tag.

An alternative method for RFID authentication is based on a “challenge and response” between a reader and a tag. Juels et. al. [58] observed that human authentication protocols can be applied to RFID, since RFID tags, like humans, have weak computational capabilities. They introduced HB protocol, in which a reader issues a new challenge to a tag each time it queries an RFID tag. The tag computes the binary inner product based on the reader’s challenge, and returns the answer to the reader. The reader authenticates the tag by verifying the tag response. The HB+ protocol is an improvement over the HB protocol by using an additional binding factor from the tag to defend against an active adversary. Later work by [22, 44, 86] improves on this idea.

YA-TRAP [106] introduces a novel technique using timestamps in RFID authentication. This is a novel approach since RFID tags have no self-contained power source to keep track of time. In YA-TRAP, a reader will send a timestamp of the current time to a tag which then decides whether to return a random reply or an encrypted reply based on the received timestamp and its own internal timestamp. The reader sends this reply back to a backend server to obtain the tag data. Chatmon et. al. [25] suggested an improvement to this protocol.

An assumption made by earlier research, is that RFID tags are capable of executing cryptographic hash functions. However, most current commercial RFID tags do not provide these hash functions, mainly due to the higher production cost [117]. A cryptographic hash function requires

additional gates to be implemented in the tag, raising the overall cost per tag. Common hash functions like MD4, SHA-1 and SHA-256 require between 7350 and 10868 additional gates [39]. This suggests that the majority of the proposed protocols are likely to be feasible only on expensive RFID tags attached to more valuable items. Recent work by [18] suggested using physically unclonable functions (PUF) in RFID tags since they only require 545 gates to implement. However, the same paper also noted that PUF-based hash functions are difficult to analyze since they are influenced by physical environment. How to design security protocols using PUF-based hash functions remain an open problem.

An orthogonal approach to RFID security focuses on changing the physical hardware of the RFID tag itself. Efforts by [11, 12, 65] investigated the possibility of building RFID hardware that is capable of performing public key based authentication. Their efforts have centered on using a particular flavor of public key cryptography based on elliptic curve cryptography (ECC). ECC has been suggested as a good replacement for RSA based public key cryptosystems since a 160-bit ECC offers the same level of security as a 1024-bit RSA encryption. While a public key cryptosystem for RFID tags greatly improves RFID privacy and security, it is also more costly to implement than cryptographic hash functions. Furthermore, it is unclear whether tiny sensor motes will be used in lieu of these RFID tags, since current sensor motes are already capable of efficiently performing ECC primitives [112, 113] and protocols [111].

## **2.2 RFID Authentication**

For RFID tags attached to personal items like a passport, exposing information from these tags to an unauthorized reader violates the privacy of the owner of the item. There are two ways information about a tag can be exposed. The first is when an unauthorized reader queries the

tag and gets back the tag data. This can be solved by encrypting the tag reply such that only an authorized reader can decrypt the reply. The second is when an unauthorized reader obtains a constant reply from an RFID tag. The unauthorized reader can use this information to track the movements of the holder of an RFID tag. For instance, consider a tag attached to a passport. An unauthorized reader queries the tag and obtains a constant encrypted reply. Even though the unauthorized reader cannot decrypt the reply, it can compare tag replies at different locations. When the same tag reply is obtained in two separate locations, the unauthorized reader can infer that the holder of the tag has been to these two locations. This is also known as violating the “location privacy” of the tag. Location privacy can be solved by having each tag reply be different and unlinkable to previous tag replies.

RFID tags are also widely used as a means of identification. For example, an RFID tag can be attached to a container of pharmaceuticals so that a reader can query the tag and learn the contents without opening up the container. An adversary manufacturing counterfeit pharmaceuticals will attempt to create a fraudulent RFID tag to place onto his container of counterfeit drugs. An RFID reader that queries and accepts the fraudulent tag as a real RFID tag will then accept the counterfeit drugs as genuine.

A basic component of RFID security is to allow a reader to distinguish a real RFID tag from a fake tag. This is accomplished by having a secret known only to a reader and a genuine tag. The RFID tag can then use this secret to prove itself to a reader. An adversary attempting to create a fraudulent tag indistinguishable from a real tag needs to obtain this secret. The adversary has three methods to try to obtain this secret. The first is by eavesdropping on the communication between a reader and a tag. The second is by repeatedly querying the RFID tag to obtain enough information to derive the secret. Finally, the adversary can physically compromise the RFID tag

to obtain the secret. In this work, we only defend against the first two methods. Tamper proof hardware capable of foiling a physical attack is beyond the scope of this work.

We present the authentication protocol in this section, and leave the evaluation to the next section. For the remainder of this chapter, we consider the data a tag transfers to a reader to be the ID of the tag.

### 2.2.1 Setup

We consider an RFID reader denoted as  $R$ . Each  $R$  has a unique identifier  $r$  and an access list,  $L$ .  $R$  obtains  $r$  and  $L$  from a certificate authority,  $CA$ , after authenticating itself. The  $CA$  is a trusted party responsible for deploying all the RFID tags and authorizing all the RFID readers. We assume that communications between  $R$  and the  $CA$  are performed via a secure channel. Subscripts are used to distinguish one reader from another. Thus RFID reader  $i$  will be  $R_i$ , with a identifier  $r_i$  and access list  $L_i$ . Each RFID tag,  $T$ , contains a unique value  $id$ , a unique secret  $t$ , knowledge of functions  $f(.,.)$  and  $h(.)$ . The  $id$  is an unique identifier for  $T$ , and is the tag data requested by a reader. The secret  $t$  is the tag secret known only by the tag itself and  $CA$ . The function  $h(.)$  is a one way hash function that outputs a bitstring of length  $l$ . A shorter length  $m < l$  is predefined by the  $CA$  and known to all readers and tags. The function  $f(.,.)$  is the hash function  $h(.)$  applied to the concatenation of two arguments. For instance, a tag  $T$  applying  $f(.,.)$  to an argument  $r$  sent by  $R$  will then have  $f(r,t) = h(r||t)$  where  $||$  denotes concatenation.

After reader  $R_i$  authenticates itself to  $CA$  and obtains access to RFID tags  $T_1 \cdots T_n$ ,  $R_i$  will have  $L_i$  where

$$L_i = \begin{cases} f(r_i, t_1) & : id_1 \\ \cdots & : \cdots \\ f(r_i, t_n) & : id_n \end{cases}$$

Note that  $R_i$  does not know any of the tags secret  $t$ . It only knows the outcome of the function

**Table 2.1:** Notations

$CA$	Trusted party, responsible for authenticating readers and deploying tags
$R_i$	RFID reader $i$
$r_i$	id for RFID reader $R_i$
$L_i$	access list for RFID reader $R_i$
$n$	number of entries in $L_i$
$T_i$	RFID tag $i$
$id_i$	id for RFID tag $T_i$
$t_i$	secret for RFID tag $T_i$
$h(x)$	one-way hash function
$f(x,y)$	Concatenate $x$ and $y$ , then applying $h(\cdot)$ , $h(x  y)$
$l$	number of bits of hash $h(\cdot)$
$m$	$CA$ defined number of bits, $m < l$

$f(r,t)$ . We assume that the  $CA$  cannot be compromised, and that all readers once authenticated by the  $CA$  are trusted. They will not reveal their access lists to anyone else. Next, we present our authentication protocol.

### 2.2.2 Authentication Protocol

$$R_i \rightarrow T_j : \text{request} \quad (2.1)$$

$$R_i \leftarrow T_j : n_j \quad (2.2)$$

$$R_i \rightarrow T_j : n_i, r_i \quad (2.3)$$

$$R_i \leftarrow T_j : h(f(r_i, t_j))_m, h(f(r_i, t_j) || n_i || n_j) \oplus id_j \quad (2.4)$$

$$R_i : \text{Hash every entry in } L_i \text{ and check} \\ \text{if first } m \text{ bits match } h(f(r_i, t_j))_m \quad (2.5)$$

$$R_i : \text{Checks } L_i \text{ for matching } h(f(r_i, t_j))_m \quad (2.6)$$

$$R_i : \text{Determine } h(f(r_i, t_j) || n_i || n_j), \text{ obtain } id_j \quad (2.7)$$

where  $n_i$  and  $n_j$  are random numbers generated by  $R_i$  and  $T_j$  respectively.  $T_j$  sends its  $id_j$  as  $h(f(r_i, t_j) || n_i || n_j) \oplus id_j$ . The tag also sends  $h(f(r_i, t_j))_m$  to help  $R_i$  reduce the time taken to search through  $L_i$ . An unauthenticated reader cannot obtain  $id_j$  since he does not know  $f(r_i, t_j)$ , and hence cannot compute the  $h(f(r_i, t_j) || n_i || n_j)$  necessary to obtain  $id_j$ . This is a form of tag authenticating reader, since the value of the tag is incomprehensible to an unauthorized reader.

The reader checks his  $L_i$  for matching entries that have the same first  $m$  bits as  $h(f(r_i, t_j))_m$ .  $R_i$  can precompute the  $h(f(r_i, t_*)_m$  for every entry in  $L_i$ , and then organize the result into corresponding groups. If there are no entries in  $L_i$  that match the first  $m$  bits, then either the RFID tag is a fake, since it is not able to generate a correct  $f(r_i, t_j)$ , or that it is a tag that  $R_i$  is not authorized to access, thus not appearing in  $L_i$ . If there is a match, the reader then uses the random numbers  $n_i$  and  $n_j$  to obtain  $h(f(r_i, t_j) || n_i || n_j)$  and the resulting  $id_j$ . If the  $id_j$  received from the tag does not match any entry in  $L_i$  then  $R_i$  ignores the tag. Note that a different random numbers  $n_j$  and  $n_i$

are used in each transaction, which means that the shared secret between  $R_i$  and  $T_j$  used to protect  $id_j$ ,  $h(f(r_i, t_j) || n_i || n_j)$ , changes each time. Also, since hash  $h(\cdot)$  is a one way hash function, even knowing the entire  $h(f(r_i, t_j))_m$  does not reveal  $f(r_i, t_j)$ .

To determine the value of  $m$ , we first define a *collision space*  $CS$  whose cardinality is  $2^{l-m}$ . This is the expected number of RFID tags whose hashed value share the same first  $m$  bits. We define  $\beta$  as the probability that, given a tag, the probability that when a reader reads in another tag having the same first  $m$  bits, the two tags are the same. The more privacy we wish, the smaller we set  $\beta$ . Thus, we have

$$\frac{\binom{CS}{1}}{CS^2} = \frac{1}{CS} = 2^{m-l} \leq \beta \Rightarrow m \leq l + \log \beta.$$

The search time for  $R_i$  becomes  $O(\frac{L}{2^m})$  since  $R_i$  can organize  $L_i$  into respective groups after  $T_j$  returns the first  $m$  bits of  $h(f(r_i, t_j))_m$ . Thus,  $R_i$  does not need to search the entire  $L_i$ , but only the smaller group of size  $\frac{L}{2^m}$ .

### 2.3 Security Analysis

In this section, we analyze our protocols against different types of attacks. For each attack, we first give a brief description of the attack, and the common assumptions about the adversary. This is followed by an explanation of how the protocols defend against the attack. We denote the adversary as  $\alpha$ , and a legitimate reader and tag as  $R_i$  and  $T_j$  respectively. A fake tag  $j$  impersonating the real tag  $j$  is depicted as  $\hat{T}_j$ .

**Basic Privacy:** The basic privacy attack occurs when  $\alpha$  wishes to learn of the content of  $T_j$ . Consider for example, the tag  $T_j$  attached to a valuable container in a warehouse. Under this attack, we generally assume that  $\alpha$  has a list of targeted RFID tag.  $\alpha$  then queries every tag in the

warehouse to decide the most valuable one to steal. In our protocol, each time any reader queries  $T_j$ ,  $T_j$  generates a new response  $h(f(r, t) || n_r || n_t)$  for authentication. Thus  $\alpha$  cannot identify which RFID tag is on his list. This protects the privacy of the tag.

**Tracking:** Under this attack,  $\alpha$  tries to track  $T_j$  over time.  $\alpha$  succeeds if he is able to distinguish  $T_j$  from other RFID tags over time. For example,  $T_j$  could be attached to a passport. By repeatedly querying with a value that yields a consistent reply,  $\alpha$  will be able to track the movements of  $T_j$  over time. This consistent reply becomes a signature of  $T_j$ .

Under our scheme,  $\alpha$  can reuse the same  $n_\alpha$  and  $r_\alpha$  for every query, but cannot predict the random  $n_j$  generated each time by  $T_j$ . In the protocol we return the entire  $h(f(r_i, t_j) || n_i || n_j)$  XORed with  $id_j$ . Since  $n_j$  is a random number chosen by the tag for each query,  $\alpha$  learns nothing from repeated queries.

Note that we also return  $h(f(r_i, t_j))_m$  in step (4) which could be used to track  $T_j$ . This is an optimization step done to improve the search time for  $R_i$ . Step (4) can be modified to return just  $h(f(r_i, t_j) || n_i || n_j) \oplus id_j$  to make tracking impossible. However, by keeping  $m$  small, the risk of tracking is minimal since there could be multiple RFID tags with the same first  $m$  bits.

**Cloning:** We consider the “skimming” attack described by Juels [56]. Under this attack,  $\alpha$  will usually first query  $T_j$  and obtain a response.  $\alpha$  then places the response on a fake RFID tag,  $\hat{T}_j$ . By creating fake RFID tags that contain the responses of real RFID tags,  $\alpha$  attempts to pass off his counterfeits as legitimate.  $\alpha$  succeeds if  $R_i$  believes that  $\hat{T}_j$  is  $T_j$ .

Under our protocol,  $T_j$  will return a different hash based on the random  $n_i$  and  $r_i$  provided by  $R_i$ . Since  $\alpha$  cannot predict the random  $n_i$  generated each time by  $R_i$ , the hash value that  $\alpha$  obtains from  $T_j$  will not be the same as the value  $R_i$  obtains when he queries  $T_j$ . Thus  $\alpha$  cannot create a  $\hat{T}_j$  that can fool  $R_i$ .



**Eavesdropping:** Here  $\alpha$  is able to observe *all* interactions between  $R_i$  and  $T_j$ . In other words, under protocol 1,  $\alpha$  learns  $r_i, n_i, n_j$ , as well as the challenge and response between the  $R_i$  and  $T_j$ . Under protocol 2,  $\alpha$  learns  $r_i, n_i, n_j, h(f(r_i, t_j)||n_i||n_j) \oplus id_j$  and  $h(f(r_i, t_j))_m$ .  $\alpha$ 's goal is to use the data to launch any of the three attacks mentioned above.<sup>1</sup>

For both protocols, every transaction between  $R_i$  and  $T_j$  begin by both parties generating a different  $n_i$  and  $n_j$ . An  $\alpha$  eavesdropping on the communication observes a different query and a different response each time, even if  $R_i$  is querying the same tag  $T_j$ . Thus, our protocols prevent  $\alpha$  from using eavesdropping to launch a basic privacy attack or tracking attack.

An  $\alpha$  can try to clone a tag by creating a fake tag with the eavesdropped information. However,  $\alpha$  cannot control the random number  $n_r$  chosen by the  $R_i$  for each new query. Under both authentication protocols, each new query generates a new hashed result  $h(f(r_i, t_j)||n_i||n_j)$ . Since  $\alpha$  does not know  $f(r_i, t_j)$ ,  $\alpha$  cannot derive the correct hash result, even if it knew what the random numbers were.

**Physical attack:** We consider two different flavors of physical attack. The first is when  $\alpha$  compromises the reader  $R_i$ . The second is when  $\alpha$  compromises the tag  $T_j$ . In both cases, we assume that once  $\alpha$  has physically compromised  $R_i$  and  $T_j$ , and  $\alpha$  will learn everything about  $R_i$  and  $T_j$ . Hardware-based defenses against physical attacks are beyond the scope of this research.

First, we consider  $\alpha$  compromising  $R_i$ .  $\alpha$  will know the contents of  $L_i$ , as well as  $r_i$ .  $\alpha$  will therefore be able to impersonate  $R_i$  and obtain data from tags  $T_1, \dots, T_n$ . The goal is to prevent  $\alpha$  from using the knowledge to create counterfeit tags. Let  $T_j$  be in  $L_i$ , and  $\alpha$  wishes to create a counterfeit tag  $\hat{T}_j$  that can fool another authenticated RFID reader  $R_x$ .  $\alpha$  knows  $f(r_i, t_j)$  and  $id_j$

---

<sup>1</sup>This version of eavesdropping is stronger since it assumes that  $\alpha$  can eavesdrop on both reader-to-tag and tag-to-reader communications. A weaker version of eavesdropping considered by some researchers assume that  $\alpha$  can only eavesdrop on the reader-to-tag communication.

from  $L_i$ . To create  $\hat{T}_j$  to fool  $T_x$ ,  $\alpha$  has to be able to derive  $f(r_x, t_j)$ . This is because each  $f(.,.)$  value in the access list is different for every RFID reader.  $R_i$  will have  $f(r_i, t_j)$ , and  $R_x$  will have  $f(r_x, t_j)$ . Thus  $\alpha$  cannot substitute his  $f(r_i, t_j)$  and  $id_j$  into  $\hat{T}_j$ . Since  $f(.,.)$  is irreversible,  $\alpha$  cannot derive  $t_j$  from  $f(r_i, t_j)$ .

Next, we consider  $\alpha$  compromising tag  $T_j$ .  $\alpha$  will now be able to create a fake  $\hat{t}_j$  that can fool the honest  $R_i$ . We want to prevent  $\alpha$  from creating another tag that can fool  $\alpha$ . We let this other tag be  $T_x$ , and assume that  $T_x$  is inside  $L_i$ . Since  $\alpha$  has compromised  $T_j$ , we assume that  $\alpha$  knows any information that  $R_i$  passes to  $T_j$ . To create  $T_x$  to fool  $R_i$ ,  $\alpha$  has to be able to generate the correct  $f(r_i, t_x)$ . However, each RFID tag has a unique secret  $t$ . Thus  $\alpha$  knowing  $t_j$  cannot derive  $t_x$ . Therefore,  $\alpha$  cannot create a fake  $T_x$  to fool  $R_i$ .

**Denial of service (DoS):** The adversary  $\alpha$  here does not try to obtain information from the tag, but rather tries to ensure that a legitimate  $R_i$  cannot access the data stored in  $T_j$ . To launch a DoS attack,  $\alpha$  sends a large number of requests to the backend server to overwhelm the server. This results in a legitimate  $R_i$  being unable to access the database to obtain information about the tag. Under our solutions, a reader only needs to contact the server once to obtain an access list  $L_i$ . The reader is then able to interact with RFID tags without further interaction with the server. A DoS attack under our schemes will not affect readers that have already been authenticated. Only readers yet to obtain an access list are affected. Thus, our serverless protocol mitigates the damage of a DoS attack.

## 2.4 RFID Search

Complex RFID operations which require data from a large collection of RFID tags usually assume that the data have already been collected and stored into a database [45,46]. Any RFID authentica-

tion protocol which provides security and privacy protection can be used. However, as the number of RFID tags increases, the cost of collecting data can be very high. More efficient methods for performing different RFID operations are needed. In this section, we consider one such operation: searching for an RFID tag from a large collection of tags. Search is a basic and invaluable tool for sifting through large amounts of data. Consider for example, a large pharmacy stocked with RFID embedded medication. A pharmacist wanting to find a particular drug can broadcast his query and receive an answer. Due to the limited broadcast range of RFID readers, the pharmacist can even determine the approximate locality of the medication by directing the RFID reader at different locations, i.e. different shelves.

Ideally, we want a reader to be able to query for a specific tag and have only that tag to reply. To illustrate, we have  $R_i$  wanting to find the tag  $T_j$ .

$$R_i \rightarrow T^* : id_j \quad (2.1)$$

$$T^* : \text{If } id = id_j \quad (2.2)$$

$$R_i \leftarrow T_j : \text{Reply} \quad (2.3)$$

where  $T^*$  refers to an arbitrary tag in the collection. However, this simple protocol does not provide any privacy or security protections. An adversary, for example, can query for valuable tags to steal. To provide security and privacy, an RFID tag should authenticate the reader before replying. Also, the RFID reader should ensure that only genuine RFID tags receive his query. This prevents an adversary from learning the content of the query. The adversary knowing the query and observing a reply, can conclude that a particular tag is in the collection, since only a tag matching the query will reply. We can thus characterize the problem as follows. Tags should only respond to authenticated readers. Readers should only query authenticated tags. This creates

a chicken-and-egg problem since **readers want to query authenticated tags, but tags will only respond to authenticated readers.**

A solution is for the reader to issue a search request such that only an authenticated tag can understand, and for the tag to reply in such a manner that only an authenticated reader can understand. An adversary can still observe all the transactions, in that he can observe there has been a query and an answer. However, since the adversary does not know the content of the query, observing the existence of an answer is not useful. For the remainder of this section, “query” and “search request” are used interchangeably. The secure search protocol is as follows.

$$R_i \rightarrow T^* : h(f(r_i, t_j) || n_r) \oplus id_j, n_r, r_i \quad (2.1)$$

$$T^* : \text{Derive } h(f(r_i, t) || n_r) \text{ and XOR with } h(f(r_i, t_j) || n_r) \oplus id_j \quad (2.2)$$

$$: \text{If } id = id_j \quad (2.3)$$

$$R_i \leftarrow T_j : h(f(r_i, t_j) || n_r || n_t) \oplus id_j, n_t \quad (2.4)$$

The search request for  $id_j$  is sent as  $h(f(r_i, t_j) || n_r) \oplus id_j$ . A tag needs to have the tag secret  $t_j$  to successfully execute step (2) and obtain  $id_j$ . Since  $\alpha$  does not know  $t_j$ , he is unable to determine what the reader is searching for. Each reader’s query is different due to the random  $n_r$  generated for each new search request. Thus, even if the reader repeatedly searches for the same tag,  $\alpha$  will obtain a different search request each time. A reader receiving a tag reply  $h(f(r_i, t_j) || n_r) \oplus id_j, n_t$  needs  $f(r_i, t_j)$  to obtain  $id_j$ , and  $f(r_i, t_j)$  is known only to the authorized reader. Thus,  $\alpha$  cannot create a fake tag  $\hat{T}_j$  to fool the reader.

### 2.4.1 Security Analysis

The security analysis in section V also applies to the search protocol with one exception, the search protocol presented above is not resistant to tracking.

Consider the following attack where  $\alpha$  eavesdrops on a transaction between a reader and a group of tags. Adversary  $\alpha$  is unable to decrypt the query or the reply, but can detect the presence of a query and reply.  $\alpha$  then broadcasts the same query repeatedly. Since the query is legitimate, the tag with the corresponding value will reply. Even though the reply is different every time due to the random  $n_t$  generated by the tag, there can only be one reply since each tag has its own unique secret  $t$ .  $\alpha$  can extend the attack by isolating each tag in the group and repeating the query, waiting for a reply.  $\alpha$  then combines this with physical observation to determine the identity of a tag.

We stress that the tracking attack presented here is different from tracking attacks commonly found in RFID security literature. The adversary cannot pick a particular tag to track. Rather, he can only track a tag which has been searched for by a legitimate reader. Furthermore, the adversary has to iteratively query every tag in a group individually before determining what tag he is tracking. These reasons increase the difficulty of launching a tracking attack via the RFID search protocol.

This attack underscores a fundamental difficulty in developing a secure search protocol for RFID tags. **The very act of replying of a query can be used to identify a tag.** So long as a search query produces a unique reply, the reply becomes an identifier for a particular tag. Encryption does not solve the problem, since encryption only prevents an adversary from learning the content of a message, but not that a message has been sent.

### 2.4.2 Search Protocol Improvements

Here we suggest several improvements to the search protocol to minimize the impact of tracking. One solution is to force the reader to use a different random number  $n_r$  for each new query. This can be accomplished by having the RFID tag store a list of random numbers used in earlier queries. When a query arrives with an  $n_r$  that appears in this list, the tag will refuse to reply. This way, an adversary will not be able to replay an eavesdropped query. An incrementing counter cannot be used by the tag to store the random numbers since a legitimate reader will generate a new random number each time. Below, we present the protocol where a tag can only remember the last used random number.

$$R_i \rightarrow T^* : h(f(r_i, t_j) || n_r) \oplus id_j, n_r, r_i \quad (2.1)$$

$$T^* : \text{Deriving } h(f(r_i, t) || n_r) \text{ and XOR with } h(f(r_i, t_j) || n_r) \oplus id_j \quad (2.2)$$

$$: \text{ If } id = id_j \text{ and } n_r \neq oldn, \text{ update } oldn = n_r \quad (2.3)$$

$$R_i \leftarrow T_j : h(f(r_i, t_j) || n_r) \oplus id_j, n_r \quad (2.4)$$

where  $oldn$  is the previous random number used. Now,  $\alpha$  cannot replay  $h(f(r_i, t_j) || n_r) \oplus id_j, n_r, r_i$  to get a reply, since  $n_r$  was just used. The adversary does not know  $f(r_i, t_j)$ , thus cannot generate his own legitimate query that will be answered by the tag. The adversary can observe the next time  $R_i$  does a search query to obtain a different random number,  $n'_r$ .  $\alpha$  can now try to use the previous search query. However, since adversary cannot determine the contents of the query, he cannot know if  $R_i$  was querying for the same tag or not. Provided that the adversary cannot determine *what*  $R_i$  is looking for, he cannot track any tag based on two reader queries. In general, an adversary will need at least one more successful query than the number of tags to be always

successfully track one tag. By the pigeonhole principal, with  $n$  tags each capable of storing the last  $m$  random numbers of successful reader query, an adversary can only guarantee to be able to track 1 tag after  $n \cdot m + 1$  queries. However, this method is ineffective against an opportunistic adversary who simply replays the overheard queries over and over again to find at least 1 tag to track.

Another solution is to adopt a challenge and response method. The idea is to avoid the condition where replying to a query can be used to identify a tag. We use  $[id_j]_m$  to denote the first  $m$  bits of  $id_j$  and  $id_m$  to denote the first  $m$  bits of a generic tag's  $id$ . The protocol is as follows.

$$R_i \rightarrow T^* : \text{Broadcast } [id_j]_m, r_i, n_r \quad (2.1)$$

$$T^* : \text{If } id_m = [id_j]_m \quad (2.2)$$

$$R_i \leftarrow T_j : h(f(r_i, t_j) || n_r || n_t) \oplus id_j, n_t \quad (2.3)$$

$$R_i : \text{Determines } f(r_i, t_j) \text{ from } L, \text{ obtain } id_j \quad (2.4)$$

Under this protocol, any tag that matches the first  $m$  bits of  $id_j$  will reply to the query. Depending on the length of  $m$ , there could be multiple tags that share the same first  $m$  bits.  $R_i$  can use existing anti-collision techniques to obtain  $id_j$ . Since multiple tags may share the same  $m$  bits,  $\alpha$  cannot infer any unique information from the reply. A tag's response is protected by the XORing their value with  $h(f(r_i, t_j) || n_r || n_t)$ . Only an authenticated reader will know  $f(r_i, t_j)$ , and be able to generate the correct hash value. Furthermore, each party contributes a random number  $n_r$  and  $n_t$  that make up the final hash value needed to successfully obtain the  $id_j$ . This prevents an adversary from launching a replay attack from either the query or reply.

This solution does not work well when the  $id$  for each tag is structured. For example, the first several bits of an  $id$  could signify general product code, the next several bits the tag origin and so

on. In this scenario, the adversary can obtain some information simply by observing  $[id_j]_m$ . Note that  $[id_j]_m$  cannot be XORed with some  $f(r_i, t_j)$  since then only  $T_j$  can decipher the request.

The last solution is to use noise to mask the reply. Each tag receiving a search query that does not match the request will have some probability of replying. Thus,

$$R_i \rightarrow T^* : \text{Broadcast } h(f(r_i, t_j) || n_r) \oplus id_j, n_r, r_i \quad (2.1)$$

$$T^* : \text{Derive } h(f(r_i, t) || n_r) \text{ and XOR with } h(f(r_i, t_j) || n_r) \oplus id_j \quad (2.2)$$

$$: \text{ If } id = id_j : \quad R_i \leftarrow T_j : h(f(r_i, t_j) || n_t) \oplus id_j, n_t \quad (2.3)$$

$$: \text{ Else : } \quad R_i \leftarrow T_j : (rand, n_t) \text{ with probability } \lambda \quad (2.4)$$

where  $\lambda$  is the predefined probability that a tag that does not match  $id_j$  will reply. Here, an adversary cannot depend on replaying a previous query to track a tag since any tag could reply. This method also avoids leaking any information to an adversary. To estimate  $\lambda$ , we first let  $S$  be the number of RFID tags that can hear a single broadcast query. We want to have a probability of  $\gamma$  that at least one tag that is not the answer to reply to create noise. We can estimate  $\lambda$  by solving  $1 - (1 - \lambda)^S \geq \gamma$ . The additional work done by reader to filter out the noise is  $O(\lambda \cdot S)$ . However, this solution only performs well when we have a reliable  $S$ , for example, a group tags are placed in a shipping container.

## 2.5 Additional Discussion

Despite the shortcomings of the central database model, it does have two advantages over a serverless solution. The first is the ease of performing revocation, and the second is fine grain access control.



The central database model provides an implicit revocation capability since the RFID reader has to contact the central database each time to obtain the tag data. To revoke a reader, the central database simply ignores the reader. Under our scheme, simple revocation can be accomplished by replacing the existing RFID tag with a new tag containing a new secret  $t$  when necessary. This solution is practical when RFID tags are passed from one owner to another. Different owners will want to attach their own RFID tags to their objects to better interface with their existing RFID management applications. An alternative revocation scheme is to retain the RFID tags, but allow the RFID tag's secret  $t$  to be changed by trusted parties. A special secret pin can be built into each RFID, and knowledge of the pin will allow the reader to change the tag secret. This pin can be transmitted directly to trusted agents of the CA, or encoded via a different channel like a 2-D barcode next to the RFID tag [56, 57]. In this way, the CA can enforce a time period in which authorized readers can access the tag data.

The other implicit advantage of the central database model is fine grain access control. When the central database returns the tag data to the reader, it can choose to only return part of the information depending on the permissions of the reader. We can provide fine grain access control in our scheme by replacing the single secret  $t$  in each RFID tag with multiple secrets depending on the granularity. For example, an RFID tag whose data consists of a general product code and unique identifier will have two secrets  $t^1, t^2$ . A reader with access to the general product code will only receive  $f(r, t^1)$  in his  $L$  while another reader with access to the unique identifier will receive  $f(r, t^2)$  as well. We can simply extend the number of secrets per tag to as fine a level of access control as desired.

Finally, we discuss cost and efficiency. For our authentication protocols, the first protocol requires the tag to perform two hash functions,  $f(.,.)$  and  $h(.,.)$ . The second protocol requires

three hash functions,  $f(.,.)$  once and  $h(.)$  twice. For the search protocols, the second search improvement requires the tag to execute two hash functions, and the remaining search improvements require three hash functions. The cost for our protocols is higher than alternative protocols [78, 106, 117] which require the tag to perform only one hash function. The additional hash functions allows our protocols to be serverless and yet avoid exposing the tag secret to the reader. Considering communication cost, the first authentication protocol requires transferring  $2 \cdot |n| + |r_i| + b + 2 \cdot |k + ans|$  bits. Assuming that both reader and tag ids have the same length, authentication protocol 2 requires  $2 \cdot |n| + 2 \cdot |id_j| + m$  bits. The communication cost for search protocols is higher since the reader's query contains of the tag id he is looking for. Again assuming both tag and reader ids have the same length. search improvement 1 transfers  $3 \cdot |id| + 2 \cdot |n|$  bits. Improvements 2 and 3 transfers  $2 \cdot |id| + m + 2 \cdot |n|$  bits and  $3 \cdot |id| + 2 \cdot |n|$  bits respectively.

In terms of efficiency, the reader  $R_i$  has to perform  $|L_i|$  hashes and search  $|L_i|$  entries for each new query under authentication protocol 1, where  $|L_i|$  is the size of the access list. For authentication protocol 2, the reader needs to perform  $|L_i|$  hashes once to derive  $h(f(r_i, t_*))$ . For each new query, the reader only performs the hash for replies that match the first  $m$  bits of  $h(f(r_i, t_*))$ , resulting on average hashing and searching  $\frac{|L_i|}{2^m}$  entries. The reader's performance for search protocols is very efficient since the reader only needs to check the access list for the entry it is looking for.

## 2.6 Conclusion

In this chapter, we present authentication and search protocols for RFID tags. Our authentication protocols provide both tag-to-reader and reader-to-tag authentication and are resistant against common RFID attacks. A major departure from the previous research is that our schemes do not

require a persistent connection to a central database. We also introduce a new problem of performing secure search for RFID tags. We examine the difficulties in designing a secure search protocol, and provide several solutions. Finally, we also consider the implicit advantages of having a central database and suggest solutions for overcoming them.

## Chapter 3

# Monitoring for Missing RFID Tags

This chapter considers local data extraction from multiple RFID tags. The popularity of RFID tags will inevitably lead to situations where there are too many tags to efficiently process individually. Here we consider applications where there is no need to collect the entire tag ID to obtain the necessary information. This chapter presents our techniques to quickly extract useful information from a large set of tags.

Retail outlets lose an estimated 30 billion dollars a year to shrinkage, of which 70% are due to administration error, vendor fraud and employee theft [51]. Inexpensive RFID technology can alleviate this problem by providing a low cost and efficient means of performing inventory control. A retailer could first attach an RFID tag to each item to be monitored. Each tag contains a unique ID which is recorded and stored on a secure server. The retailer then deploys an RFID reader to periodically collect all the IDs from the tags and match them against the IDs stored on server. This way, the retailer can be immediately notified of any errors. We term this simple approach *collect all*.

However, collect all suffers from three drawbacks. First, collecting tag IDs is time consuming when there are a lot of tags due to presence of collisions. A reader collects IDs by first broadcasting

the number of available time slots. Each tag will independently pick a time slot to reply. When multiple tags pick the same slot, a collision occurs and the reader obtains no information and must repeat the process again. When the set of tags is large, the number of collisions will rise, increasing the data collection time.

Second, routine monitoring usually does not require every ID to be accounted for. Consider an RFID tag attached to every product in a grocery store, and the store contains hundreds of thousands of items. In this setting, it is impractical to notify the retailer each time there is a *single* RFID tag missing. This is because a missing ID might indicate a scratched RFID tag, or simply that the RFID tag is physically blocked from receiving the query by another object. A more reasonable approach is to determine a threshold or tolerance for missing items, and alerting the retailer only when this threshold is breached.

Third, collect all is vulnerable to dishonest RFID readers returning incorrect information to the server. This is a serious threat since an estimated 45% of thefts are committed by employees [51]. A dishonest employee can first collect all the tag IDs prior to the theft, and then replay the data back to the server later.

In this chapter, we consider the problem of accurately and efficiently monitoring for missing RFID tags. We assume that the RFID reader interacts with the tags and passes the collected information to the server. The server has preprogrammed threshold, and will issue a warning if the number of missing tags exceed the threshold. We provide two protocols, a trusted reader protocol (TRP) and an untrusted reader protocol (UTRP). The UTRP defends against a dishonest reader from returning inaccurate data to the server. Our contributions in this chapter are as follows.

- We propose a monitoring technique which does not require the reader to collect IDs from each RFID tag, but is still able to accurately monitor for missing tags.

- Our monitoring technique provides privacy protection by neither broadcasting tag IDs in public, nor revealing IDs to the RFID reader.
- We present a lightweight solution to the dishonest reader problem that does not require expensive tag hardware such as an accurate on-chip timer or cryptographic MAC functions which are unavailable on passive RFID tags.
- Our technique is more flexible than prior research in that we can accommodate different size groups of tags.

### 3.1 Related Work

In an RFID system, a collision occurs when multiple tags try to transmit data to a reader at the same time. This results in the reader being unable to obtain any useful information. Prior work [21, 24, 68, 77, 98, 109] have focused on improving protocols to reduce collisions, and secure search techniques to isolate particular tags [103] one at a time. While these techniques improve monitoring performance, such solutions are ultimately bounded by the number of tags. Regardless of the protocol used, the RFID reader will still have to isolate each tag at least once to obtain data. Our approach does not require the reader to isolate every tag.

Another approach is to use probabilistic techniques to determine some features of a large collection of RFID tags. These include methods to estimate the cardinality of a set of tags [63], and to determine popular categories of tags [97]. Our research differs from these work by including a security protocol that deals with dishonest RFID readers.

The problem of a dishonest reader is similar to the “yoking proof” problem [54, 84, 87, 95]. A yoking proof allows an RFID reader to prove to a verifier that two RFID tags were scanned

simultaneously at the same location. The yoking proof only relies on a trusted server and not a trusted RFID reader. A dishonest reader cannot tamper with the result without being detected by the sever. Bolotnyy and Robins [17] improves on the idea by creating yoking proofs for multiple tags. However, their approach requires each tag to be contacted individually *and* in a specific order. These requirements are time consuming when there are many tags. Furthermore, their scheme requires each RFID tag to have an on-chip timer that is specific to the size of the group of tags. This makes their approach inflexible in accommodating different group sizes.

### 3.2 Problem Formulation

We assume that a server has a group of objects, and an RFID tag with a unique ID is attached to each object. We refer to this group of objects as a *set of tags*. A set of tags once created is assumed to remain static, meaning no tags are added to or removed from the set.

We consider an RFID reader,  $R$ , and a set of  $n$  RFID tags,  $T_*$ . We consider this set of tags to be “*intact*” if all the tags in the set are physically present together at the same time. There are two additional parameters in our problem, a tolerance of  $m$  missing tags and a confidence level  $\alpha$ . A set is considered intact if there are  $m$  or less tags missing. The set is considered not intact where there are at least  $m + 1$  missing tags. The confidence level  $\alpha$  specifies the lower bound of the probability that a not intact set of tags is detected. Both  $m$  and  $\alpha$  parameters are set according to the server’s requirements. A higher tolerance ( $m$ ) and lower confidence level ( $\alpha$ ) will result in faster performance with less accuracy. Table 3.1 summerizes the remaining notations.

**Anti-collision :** In this chapter, we assume that RFID tags resolve collisions using a slotted ALOHA type scheme [63, 109]. The reader first broadcasts a frame size and a random number,  $(f, r)$ , to all the tags. Each RFID tag uses the random number  $r$  and its ID to hash to a slot number

$sn$  between  $[1, f]$  to return their ID, where

$$sn = h(ID \oplus r) \bmod f.$$

Tags that successfully transmit their data are instructed to keep silent. Tags that pick the same slot to reply will be informed by the reader to retransmit in subsequent rounds where the reader will send a new  $(f, r)$ . The reader repeats this process until all IDs are collected.

**Protocol goals :** The goal of a server is to remotely, quickly, and accurately determine whether a set of tags is intact. The server first specifies a tolerance of  $m$  missing tags and a confidence level  $\alpha$ , and instructs a reader to scan all the tags to collect a bitstring. The server then uses this result to determine whether there are any missing tags. Our protocols succeed if the server is able to determine a set of tags is not intact when more than  $m$  tags are missing with probability of at least  $\alpha$ . In this chapter, we assume that an adversary will always steal  $m + 1$  tags, since for any  $m$ , the *hardest scenario for the server to detect is when there are just  $m + 1$  tags missing.*

**Adversary model :** The goal of the adversary is to steal RFID tags. The adversary launches the attack by physically removing tags from the set. We do not consider more involved attacks such as “clone and replace”. In such an attack, the adversary steals some tags, clones the stolen tags to make replicate tags, and replaces the replicate tags back into the set. Cloning creates replicate tags that are identical to the stolen tags. In this scenario, the server cannot detect any missing tags since the replicate tags are identical to the removed tags. This attack requires considerable technical expertise due to the cloning process, and is unlikely to be used against commodity items tracked by low cost tags.

Our work considers two scenarios: an honest reader and a dishonest reader scenario. In the first scenario, the adversary simply attempts to steal some tags. Once the tags are stolen, the tags



are assumed to be out of the range of the reader. Therefore, when a reader issues a query, the stolen tags will not reply.

In the second scenario, the adversary controls the RFID reader responsible for replying to the server. The terms “adversary” and “dishonest reader” are used interchangeably in this chapter. After stealing some RFID tags, the adversary is assumed to be able to communicate with the stolen tags. This can be thought of as the adversary having a collaborator also armed with an RFID reader and the stolen tags. The adversary can communicate with the collaborator using a fast communication channel to obtain data about the stolen tags if needed.

**Table 3.1:** Notations

$R/T_*$	RFID reader / set of RFID tags
$f/r$	frame size / random number
$n/m$	# of tags in $T_*$ /# of tolerated missing tags
$\alpha$	confidence level
$h(\cdot)$	hash function
$sn$	slot number between $[1, f]$
$bs$	bitstring of length $f$
$c$	number of adversary communications
$ct$	counter built into RFID tag

### 3.3 TRP: Trusted Reader Protocol

In this section, we present our trusted reader protocol, TRP, where the RFID reader is assumed to be always honest. Given a set of RFID tags, TRP returns a bitstring to the server to check if the set of tags is intact.

### 3.3.1 Intuition and assumptions

TRP modifies the slot picking behavior used in *collect all* so that instead of having a tag pick a slot and return its ID, we let the tag simply reply with a few random bits signifying the tag has chosen that slot. In other words, instead of the reader receiving

$$\{\dots | id_1 | 0 | id_6 | collision | 0 | \dots\},$$

where 0 indicates no tag has picked that slot to reply, and *collision* denotes multiple tags trying to reply in the same slot, the reader will receive

$$\{\dots | random\ bits | 0 | random\ bits | collision | 0 | \dots\}.$$

This is more efficient since the tag ID is much longer than the random bits transmitted. From the reply, the reader can generate the bitstring

$$bs = \{\dots | 1 | 0 | 1 | 1 | 0 | \dots\}.$$

where 1 indicates at least one tag has picked that slot.

TRP exploits the fact that a low cost RFID tag picks a reply slot in a deterministic fashion. Thus, given a particular random number  $r$  and frame size  $f$ , a tag will always pick the same slot to reply. Since the server knows all the IDs in a set, as well as the parameters  $(f, r)$ , the server will be able to determine the resulting bitstring for an intact set ahead of time. The intuition behind TRP is to let the server pick a  $(f, r)$  for the reader to broadcast to the set of tags. The server then compares the bitstring returned by the reader with the bitstring generated from the server's records. A match will indicate that the set is intact.

### 3.3.2 TRP algorithm

The reader uses a different  $(f, r)$  pair each time he wants to check the intactness of  $T_*$ . The server can either communicate a new  $(f, r)$  each time the reader executes TRP, or the server can issue a list of different  $(f, r)$  pairs to the reader ahead of time.

Alg. 1 shows the overall interaction between the reader and tags. Each tag in the set executes Alg. 2 independently. The reader executes Alg. 3 to generate the bitstring  $bs$  and return it to the server. Notice that unlike the *collect all* method which requires several rounds to collect the tag information, our TRP algorithm only requires a single round. Furthermore, in Alg. 2 Line 5 the tag does not need to return the tag ID to the reader, but a much shorter random number to inform the reader of its presence. This shortens the transmission time since less bits are transmitted.

---

#### Algorithm 1 Interaction between tags and $R$

---

- 1: Reader broadcasts  $(f, r)$  to all tags  $T_*$
  - 2: Each tag  $T_i$  executes Alg. 2
  - 3: Reader executes Alg. 3
  - 4: Reader returns  $bs$  to server
- 

---

#### Algorithm 2 Executed by Tag $T_i$

---

- 1: Receive  $(f, r)$  from  $R$
  - 2: Determine slot number  $sn = h(id_i \oplus r) \bmod f$
  - 3: **while**  $R$  broadcasts slot number **do**
  - 4:   **if** broadcast matches  $sn$  **then**
  - 5:     Return random number to  $R$
  - 6:   **end if**
  - 7: **end while**
-

---

**Algorithm 3** Executed by Reader  $R$ 


---

```

1: Create bitstring  $bs$  of length  $f$ , initialize all entries to 0
2: for slot number  $sn = 1$  to  $f$  do
3:   Broadcast  $sn$  and listen for reply
4:   if receive reply then
5:     Set  $bs[sn]$  to 1
6:   end if
7: end for

```

---

### 3.3.3 Analysis

We present the analysis of how to choose a frame size  $f$  subject to a tolerance level  $m$  and confidence level  $\alpha$ . As mentioned earlier, we define a tolerance of  $m$  missing tags, where a set of tags can be considered intact when there are at most  $m$  missing tags from the set. The set is considered not intact when at least  $m + 1$  tags are missing. Since an appropriate value of  $m$  is application specific, we assume that  $m$  is a given parameter.

To quantify accuracy, we introduce a confidence parameter  $\alpha$ . The parameter  $\alpha$  describes the requirement of the probability of detecting at an set that is not intact. An appropriate value of  $\alpha$  is also defined by the application. A server requiring strict monitoring can assign  $m = 0$  and  $\alpha = 0.99$  for high accuracy.

Our problem can be defined as given  $n, m$  and  $\alpha$ , we want to pick the smallest  $f$  for Alg. 1 such that we can detect with more than  $\alpha$  probability when there are more than  $m$  out of  $n$  tags are missing. We use  $g(n, x, f)$  to denote the probability of detecting the set is not intact with frame size  $f$  when exactly  $x$  tags are missing. Since the scanning time is proportional to the frame size  $f$ , our problem is formulated as to

$$\begin{aligned}
& \text{minimize } f \\
& \text{s.t. } \forall x > m, g(n, x, f) > \alpha.
\end{aligned} \tag{3.1}$$

**Theorem 3.1** Given  $n, x$  and  $f$ ,

$$g(n, x, f) = 1 - \sum_{i=0}^f \binom{f}{i} p^i (1-p)^{f-i} \cdot \left(1 - \frac{i}{f}\right)^x,$$

where  $p = e^{-\frac{n-x}{f}}$ .

**Proof:** Let  $N_0$  represent the number of empty slots in the frame generated by the currently present  $n-x$  tags. A missing tag will be detected if it selects one of these  $N_0$  slots to respond, which has a probability of  $\frac{N_0}{f}$ . The probability that we can not detect any of  $x$  missing tags is  $(1 - \frac{N_0}{f})^x$ . For each slot, the probability of being one of the  $N_0$  empty slot is  $p = (1 - \frac{1}{f})^{n-x} = e^{-\frac{n-x}{f}}$ . Thus,  $N_0$  is a random variable following a binomial distribution. For  $i \in [0, f]$ ,

$$\Pr(N_0 = i) = \binom{f}{i} p^i (1-p)^{f-i}.$$

Therefore,

$$\begin{aligned}
g(n, x, f) &= 1 - \sum_{i=0}^f \Pr(N_0 = i) \cdot \left(1 - \frac{i}{f}\right)^x \\
&= 1 - \sum_{i=0}^f \binom{f}{i} p^i (1-p)^{f-i} \cdot \left(1 - \frac{i}{f}\right)^x.
\end{aligned}$$

■

**Lemma 3.1** Given  $n$  and  $f$ , if  $x_1 > x_2$ , then  $g(n, x_1, f) > g(n, x_2, f)$ .

**Proof:** It is obvious that more missing tags tend to yield higher probability of being detected. ■

**Theorem 3.2** *If we set  $g(n, m + 1, f) > \alpha$ , the accuracy constraint (3.1) is satisfied.*

**Proof:** According to Lemma 3.1,  $\forall x > m, g(n, x, f) \geq g(n, m + 1, f)$ . Therefore, missing exactly  $m + 1$  tags is the worst case for our detection. Thus, any value of  $f$  satisfying  $g(n, m + 1, f) > \alpha$  can guarantee the accuracy requirement. ■

Considering the objective, the optimal value of  $f$  is

$$f = \min\{x | g(n, m + 1, x) > \alpha\}. \quad (3.2)$$

### 3.4 UTRP: UnTrusted Reader Protocol

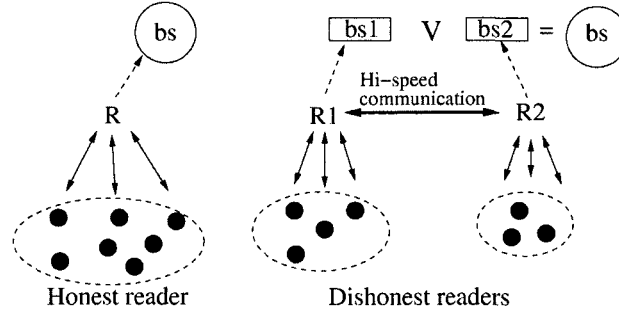
In this section, we discuss UTRP, our protocol to defend against an untrusted reader. UTRP prevents a dishonest reader from generating a  $bs$  that can satisfy the server without having an intact set. For sake of brevity, the terms “dishonest reader” and “reader” are used interchangeable for the remainder of this section. An honest reader will be explicitly specified.

#### 3.4.1 Vulnerabilities

In the introduction, we mentioned that a dishonest reader can replay a previously collected bitstring  $bs$  back to the server. This attack can be easily defeated by letting the server issue a new  $(f, r)$  each time the reader scans the set of tags. This renders previously collected bitstrings invalid. However, simply issuing a new  $(f, r)$  cannot defend against a dishonest reader and a collaborator.

The dishonest reader first steals a subset of tags from the original set of tags, and gives the stolen tags to his collaborator. We denote the remaining set of tags as  $s_1$  and the stolen tags as  $s_2$ . The collaborator is also equipped with an RFID reader. The dishonest reader is denoted as  $R_1$

and the collaborator's reader is denoted as  $R_2$ . When the server issues a new  $(f, r)$ , the dishonest reader will scan the remaining set of tags  $s_1$ , and instruct his collaborator to scan the stolen tags  $s_2$  and return the collected information. The dishonest reader will then combine the information to return a bitstring to the server. Fig. 3.1 illustrates the attack.



**Figure 3.1:** Vulnerability of TRP

The reader succeeds if he is able to generate a proof  $\hat{bs}$  from  $s_1$  and  $s_2$  located in two separate locations, such that  $\hat{bs}$  is the same as  $bs$ . The reader assigns himself as  $R_1$  to read  $s_1$  and his collaborator as  $R_2$  to read  $s_2$ . We assume that  $R_1$  and  $R_2$  both know  $(f, r)$ . Alg. 4 presents the algorithm of the attack. We see that so long as the both readers  $R_1$  and  $R_2$  have a high speed communication, they behave just like a single reader.

---

**Algorithm 4** Attack algorithm against TRP

---

- 1: Both  $R_1$  and  $R_2$  execute Alg. 1 on  $s_1$  and  $s_2$ , and obtains  $bs_{s_1}$  and  $bs_{s_2}$  respectively.
  - 2:  $R_2$  forwards  $bs_{s_2}$  to  $R_1$ .
  - 3:  $R_1$  executes  $(bs_{s_1} \vee bs_{s_2})$  to obtain  $\hat{bs}$ , where  $\hat{bs} = bs$
  - 4:  $R_1$  returns  $\hat{bs}$  to the server.
- 

One possible defense against the attack is to require a reader to complete Alg. 1 within some specified time limit  $t$ . However, selecting an appropriate value of  $t$  is difficult since  $t$  has to be long enough for an honest reader to complete a  $bs$  for the server, yet short enough such that  $R_1$

and  $R_2$  cannot collaborate by passing data to each other. For instance, in Alg. 4,  $R_1$  and  $R_2$  can derive a correct  $\hat{bs}$  by just having *one* transmission. Assuming that  $R_1$  and  $R_2$  communicates via a high speed channel, estimating a time limit  $t$  that is shorter than the time needed for a single transmission is difficult.

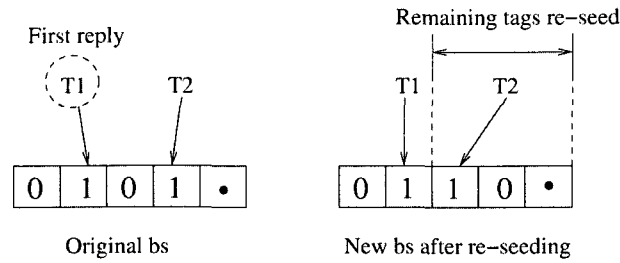
### 3.4.2 Intuition and assumptions

The intuition behind our solution is to force collaborating readers to communicate multiple times such that the latency is large enough to be accurately estimated by the server. UTRP accomplishes this by introducing two additional components, a re-seeding process, and a counter.

Each time a reader encounters a time slot that is chosen by at least one RFID tag, UTRP requires the reader to *re-seed* by sending a new  $(f, r)$  to all tags that have yet to reply. The new  $f$  is equal to the number of slots left from the previous  $f$ . For example, initially we have  $f = 10$  and the first slot has a tag reply. The new  $f$  value will thus be 9. The new random number  $r$  is determined by the server. The re-seeding will result in a  $bs$  different from the prior one. We illustrate an example in Fig. 3.2. We let the tag  $T1$  to be the first tag to reply. The reader will send a new  $(f, r)$  to remaining tags to pick a new slot. Tag  $T2$  picks a different slot after re-seeding, creating a different  $bs$ . Collaborating readers wanting to obtain  $\hat{bs} = bs$  have to re-seed each time either reader receives a reply. Since no reader can determine in advance which slot will receive a reply, collaborating readers must check with each other after either reader obtains a reply in a single slot.

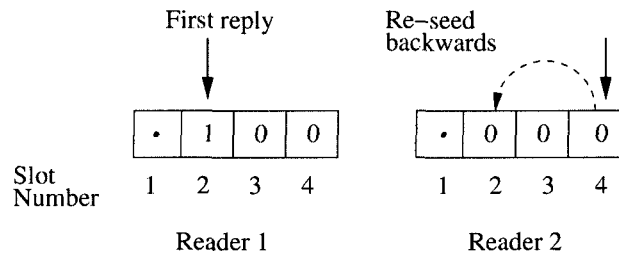
However, re-seeding does not prevent readers from running the algorithm multiple times to gain some information. Each reader can first read every slot in frame size  $f$  to determine which slot has a reply. The readers then exchange this information and scan all the tags again to arrive at





**Figure 3.2:** Re-seeding after first reply

the correct bitstring. For example in Fig. 3.3,  $R_1$  and  $R_2$  first scan all their tags to determine that a re-seed is necessary in slot 2. Both readers can then repeat the process by re-seeding tags starting from slot 2 to complete the *bs*. A mechanism to prevent a reader from going backwards is needed.



**Figure 3.3:** Re-seeding just from slot 2

We adopt an assumption made in several earlier research [17, 54, 84, 87, 95] that each RFID tag has a counter  $ct$ , and the counter will automatically increment each time the tag receives a new  $(f, r)$  pair. A reader that attempts to move backwards to re-seed the tags will have an incorrect counter value. An RFID tag now picks a slot as

$$sn = h(ID \oplus r \oplus ct) \bmod f.$$

Recall that the server knows the ID, and provides the frame size  $f$  and random numbers  $r$ . The server also knows the value of each tag's counter  $ct$  since  $ct$  only increments when queried by the

reader. Thus, the server can still determine the correct  $bs$  for verification.

---

**Algorithm 5** Interaction between server and  $R$

---

- 1: Server generates  $(f, r_1, \dots, r_f)$ , sends to  $R$ , and starts the timer
  - 2:  $R$  broadcasts  $(f, r_1)$  to all the tags  $T_*$
  - 3:  $T_*$  executes Alg. 7
  - 4:  $R$  executes Alg. 6
  - 5: **if**  $R$  returns correct  $bs$  to server before timer expires **then**
  - 6:     Server verifies  $R$ 's proof
  - 7: **end if**
- 

### 3.4.3 UTRP algorithms

We let the server issue a frame size together with  $f$  random numbers,  $(f, r_1, \dots, r_f)$ , to a reader. The reader is supposed to use each random number only once in the given order. For example, let  $f = 15$  and  $r_1 = 5, r_2 = 9$ . Reader  $R$  will first send out  $(15, 5)$ . Assuming that some tag replies in the first slot,  $R$  is supposed to re-seed by broadcasting  $(14, 9)$  so that each remaining tag can pick a new slot. A reader that does not follow this rule will not yield the right answer to the server.

Alg. 5 illustrates the overall protocol, and Alg. 6 and Alg. 7 show the reader and tag behavior respectively. Collaborating readers will have to communicate with each other after Alg. 6 Line 5 to determine whether to re-seed. If either collaborating reader receives a reply, both readers must re-seed. A reader cannot predict in advance whether any tag will reply in the next slot since a tag picks a slot number  $sn$  using the random number  $r$ , and the list of random numbers is determined by the server.

The reader also cannot attempt to execute Alg. 6 multiple times to determine which slots will

---

**Algorithm 6** UTRP algorithm for reader  $R$ 


---

- 1: Create a bitstring  $bs$  of length  $f$ , initialize all entries to 0.
  - 2: Set  $f' = f$
  - 3: **for** slot number  $sn = 1$  to  $f$  **do**
  - 4:   Broadcast  $sn - f + f'$  and listen for reply
  - 5:   **if** receive reply **then**
  - 6:     Set  $bs[sn]$  to 1, and  $f' = f - sn$
  - 7:     Broadcast  $(f', r)$  where  $r$  is the next random number in the sequence
  - 8:   **end if**
  - 9: **end for**
  - 10: Return  $bs$  to server
- 

have a reply since the counter value will change. In Alg. 7 Line 1, the tag will automatically increment the counter each time it receives a new  $(f, r)$ . Since a tag in UTRP picks a new slot using  $ID \oplus r \oplus ct$ , a different  $ct$  will cause the final  $bs$  to be different. Since an RFID tag can only communicate with a single reader at a time, the counter in Alg. 7 will not be incremented by any other readers.

### 3.4.4 Analysis

The analysis for UTRP is similar to the TRP analysis presented earlier. The difference is that in TRP, the information contained in the missing tags is gone. In UTRP, we consider the dishonest  $R$  removes more than  $m$  missing tags, but yet is able to obtain some information from the removed tags. Compared with TRP, when the same number of tags are missing, the dishonest reader has higher probability to pass the verification since the dishonest reader has more information than

---

**Algorithm 7** UTRP algorithm for tag  $T_i$ 


---

- 1: Receive  $(f, r)$  from  $R$ . Increment  $ct = ct + 1$ .
  - 2: Determine slot number  $sn = h(id_i \oplus r \oplus ct) \bmod f$
  - 3: **while**  $R$  is broadcasting **do**
  - 4:   **if**  $R$  broadcasts slot number and slot number matches  $sn$  **then**
  - 5:     Return random number to  $R$ , keep silent
  - 6:   **else if**  $R$  broadcasts a new frame size and random number  $(f, r)$  **then**
  - 7:     Receive  $(f, r)$  from  $R$ . Increment  $ct = ct + 1$
  - 8:     Determine new slot number  $sn = h(id_i \oplus r \oplus ct) \bmod f$
  - 9:   **end if**
  - 10: **end while**
- 

that in TRP.

UTRP requires the reader to return  $bs$  before timer  $t$  expires. The intuition here is to limit the communication between dishonest readers, thus increase the probability of detecting the missing tags. The communication time increases with the number of readers an adversary controls, making it easier for an adversary to be detected. In our analysis, we consider the best case for an adversary to escape detection by having the adversary only control *two* readers.

For a given frame size and random number, the scanning time for a honest reader to finish the protocol may vary. The server sets the timer to an empirical value, which is conservative so that a honest reader can definitely respond before the due time. We assume that the server can estimate the minimum and maximum scanning time of a honest reader, indicated as  $ST_{min}$  and  $ST_{max}$  respectively. The server thus sets  $t = ST_{max}$ .

Since a reader cannot predict in advance in which slot there will be a reply, UTRP forces the

dishonest readers to wait for a message from other readers every time it encounters an empty slot. If a dishonest reader receives a reply in the current slot, it can continue re-seeding and scanning the following slots without waiting for the results from other readers. We let  $t_{comm}$  be the average communication overhead between two dishonest readers. Given  $t$ , we claim that the dishonest readers can communicate in at most  $c = \frac{t - ST_{min}}{t_{comm}}$  slots.

Let us consider the whole set of  $n$  tags is divided into two sets  $s_1$  and  $s_2$ . Without loss of generality, let  $|s_1| \geq |s_2| > m$ . Assume there are two dishonest readers  $R_1$  and  $R_2$  scanning  $s_1$  and  $s_2$  respectively. Each time  $R_1$  encounters an empty slot (a slot where no tag replies),  $R_1$  will have to pause to check with  $R_2$ . If  $R_2$  receives a reply in *that* particular slot, both  $R_1$  and  $R_2$  will have to re-seed. Otherwise  $R_1$  can continue broadcasting the remaining slots. Since the dishonest readers cannot communicate after every slot within  $t$ , the best strategy for the dishonest readers to pass our verification is as follows: (1)  $R_1$  waits for the messages from  $R_2$  in the first  $c$  empty slots it has encountered; (2)  $R_1$  finishes scanning the following slots (with  $s_1$ ) and sends the bitstring to the server.

With this strategy, the first part (with communication) of the bitstring is correct, but the remaining part may be suspicious. The following analysis tries to derive an appropriate value for  $f$ , such that the server can catch the difference in this scenario with high probability ( $> \alpha$ ).

Similar to the TRP analysis, the worst case occurs when the number of missing tags is just beyond the tolerant range, i.e.,  $|s_2| = m + 1$ . Intuitively, while the number of missing tags is smaller, we need longer frame size to guarantee the same accuracy requirement. In the following, we will discuss how to set parameter in this worst case to satisfy the accuracy requirement. The optimal frame size for the worst case is thus the optimal for all cases.

**Theorem 3.3** *Assume after  $c'$  slots, the dishonest read  $R_1$  will have encountered  $c$  number of*

empty slots. The expected value of  $c'$  is  $\frac{c}{e^{-\frac{c}{n-m-1}}}$ .

**Proof:** For each slot, the probability that no tags respond is  $p = (1 - \frac{1}{f})^{|s_1|} = e^{-\frac{|s_1|}{f}}$ . After  $c'$  slots, the expected number of empty slots is  $p \cdot c'$ . By resolving  $p \cdot c' = c$ , we have  $c' = \frac{c}{e^{-\frac{c}{n-m-1}}}$ . ■

**Theorem 3.4** Let  $x$  be the number of the tags in  $s_2$ , which respond after the first  $c'$  slots. Given  $i \in [0, m+1)$ ,

$$Pr(x = i) = \binom{m+1}{i} \left(1 - \frac{c'}{f}\right)^i \left(\frac{c'}{f}\right)^{m+1-i}.$$

**Proof:** Since each tag randomly picks a slot in the frame, it has  $1 - \frac{c'}{f}$  probability to respond after the first  $c'$  slots. Thus,  $x$  follows a binomial distribution as  $x \sim B(1 - \frac{c'}{f}, |s_2|)$ . Thus, we have

$$Pr(x = i) = \binom{m+1}{i} \left(1 - \frac{c'}{f}\right)^i \left(\frac{c'}{f}\right)^{m+1-i}.$$

■

With similar proof, we have the following theorem.

**Theorem 3.5** Let  $y$  be the number of the tags in  $s_1$ , which respond after the first  $c'$  slots. Given  $i \in [0, n-m-1)$ ,

$$Pr(y = i) = \binom{n-m-1}{i} \left(1 - \frac{c'}{f}\right)^i \left(\frac{c'}{f}\right)^{n-m-i-1}.$$

On one hand, in  $s_2$ , the tags replying after the first  $c'$  slots are 'real' missing tags in this problem. On the other hand, among the tags in  $s_1$ , only those responding after the first  $c'$  slots are considered useful in detecting the missing tags. For a given frame size  $f$ ,  $f - c'$  is the effective frame size for distinguishing the bitstring with missing tags. Thus, the server has  $g(x+y, x, f - c')$  probability to detect the difference. Considering all possible values of  $x$  and  $y$ , a frame size  $f$  can satisfy the accuracy requirement, if

$$\sum_{i=0}^{m+1} \sum_{j=0}^{n-m-1} Pr(x = i) \cdot Pr(y = j) \cdot g(i + j, i, f - c') > \alpha. \quad (3.3)$$

Therefore, the optimal frame size is the minimal value satisfying the above condition.

### 3.5 Evaluation

In this chapter, we use simulations to evaluate the efficiency and accuracy of TRP and UTRP. We measure efficiency by the frame size  $f$ . A smaller  $f$  has fewer slots, which translates into faster performance. We assume the duration of each slot is equally long. We measure accuracy by first setting values of  $m$  and  $\alpha$  to derive a  $f$  satisfying Eq. (3.2) for TRP and Eq. (3.3) for UTRP. We then execute our simulation to test if our protocols can determine “missing” when there are  $m + 1$  tags randomly removed from the set. We average the results over 1000 trials.

We perform simulations varying  $n$  from 100 to 2000 tags at 100 tag increments. The tolerance level is set to tolerate  $m = 5, 10, 20$  and 30 missing tags. Finally, we uniformly set our confidence  $\alpha = 0.95$ .

Fig. 3.4 compares the efficiency between the *collect all* method against our TRP algorithm. Lee et. al. [68] determined that the optimal frame size is equal to the number of unidentified tags in a set. Based on this, we simulate *collect all* by setting  $f = n$  in the first round, and  $f$  equal to the remaining tags that have yet to transmit. The final number of slots for *collect all* method is the sum of all the  $f$ s used in each round. To accommodate the tolerance  $m$ , we consider *collect all* algorithm to be completed once  $n - m$  tags are collected. From Fig. 3.4, we observe that the scanning time in both *collect all* and TRP increases linearly as the number of tags increases. TRP uses fewer slots, especially when the set size is large. Note that the actual performance of *collect all* will be worse since the tag needs to return its ID rather than a shorter random number in TRP, resulting in a longer duration of each slot.

Fig. 3.5 shows the accuracy of TRP when using the frame size  $f$  shown in Fig. 3.4. With a

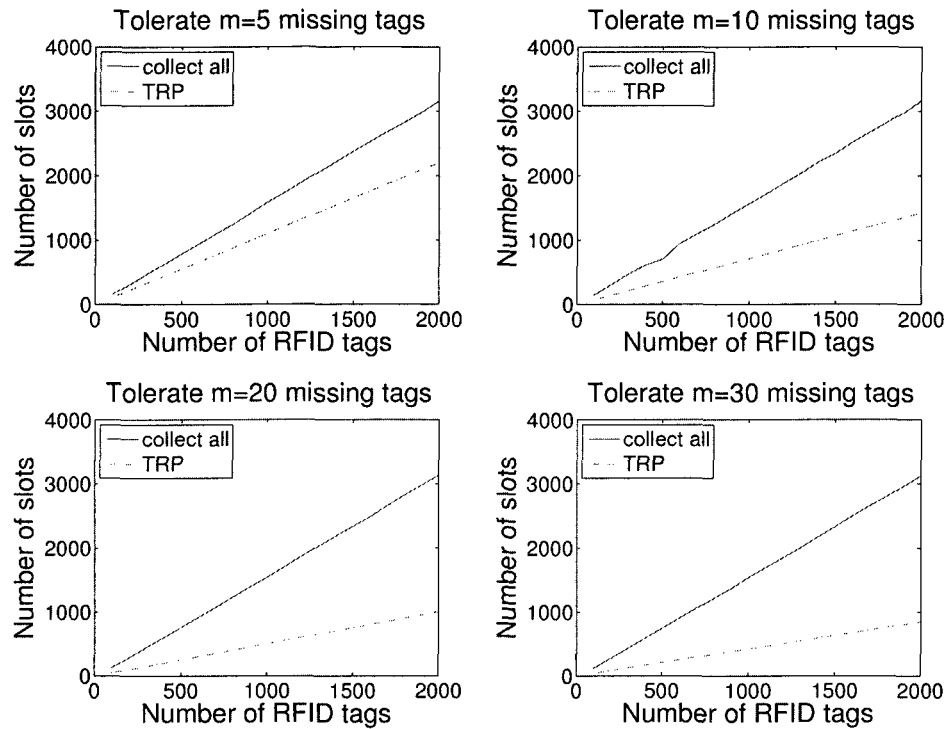
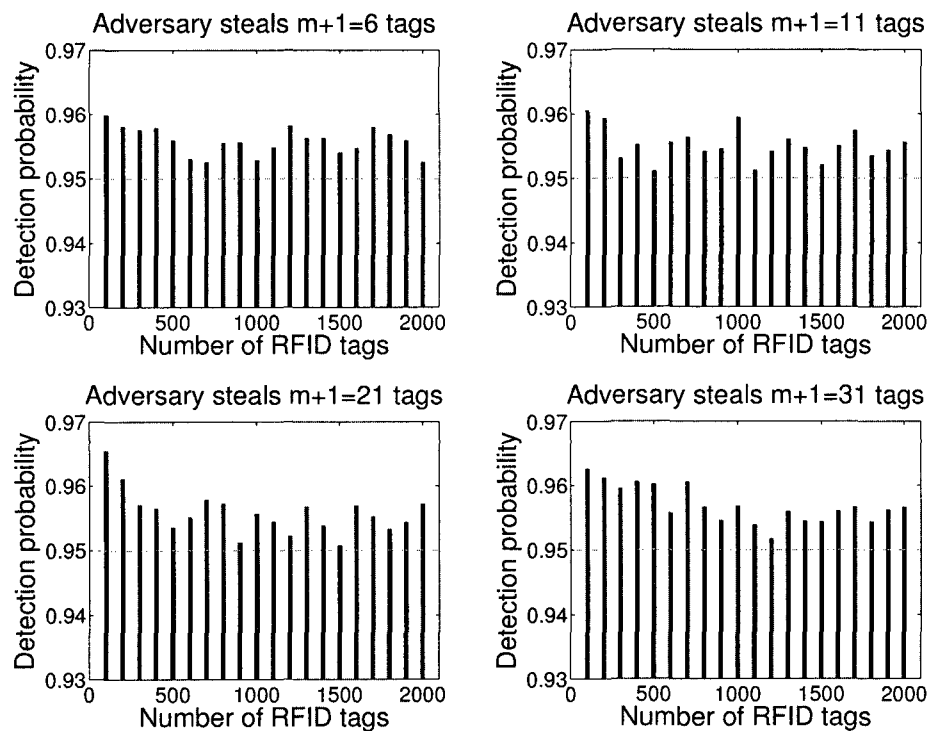


Figure 3.4: Comparing *collect all* versus TRP

tolerance of  $m$ , the most difficult situation for TRP to detect missing tags is when there are just  $m + 1$  missing tags. The horizontal dashed line in Fig. 3.5 denotes the confidence level  $\alpha$ . Each bar represents probability TRP detects  $m + 1$  missing tags from a set. Bars over the horizontal line denotes TRP has successfully detected  $m + 1$  missing tags with probability greater than  $\alpha$ . As we can see TRP detects the missing tags over probability  $\alpha$ .

In evaluating UTRP, we assume that a dishonest reader splits the set into two, and can communicate for  $c = 20$  slots before executing Alg. 5 on the remaining tags in his set. To determine the efficiency of UTRP, we compare the size of  $f$  used in UTRP against TRP, and Fig. 3.6 shows the results. We observe that the overhead of UTRP over TRP is small. Note that for UTRP, we have added a very small number of slots (between 5 to 10 slots) to the optimal frame size given





**Figure 3.5:** Accuracy of TRP with  $\alpha = 0.95$

in Eq. (3). This is because the derivation of  $c'$  in Theorem 3 relies on the expected value, which introduces a slight inaccuracy. Note that Fig. 3.6 does not imply that the performance of UTRP is comparable to TRP since we do not take into account the time needed for  $R$  to broadcast a new  $(f, r)$  pair to remaining RFID tags in UTRP. Finally, the accuracy of UTRP is shown in Fig. 3.7. UTRP also accurately detects missing tags with probability larger than the confidence level  $\alpha$ .

### 3.6 Conclusion

In this chapter, we consider the problem of monitoring for missing RFID tags. We provide protocols for both an honest and dishonest RFID reader. Our approach differs from prior work in that our techniques do not require the reader to collect the ID from every tag.

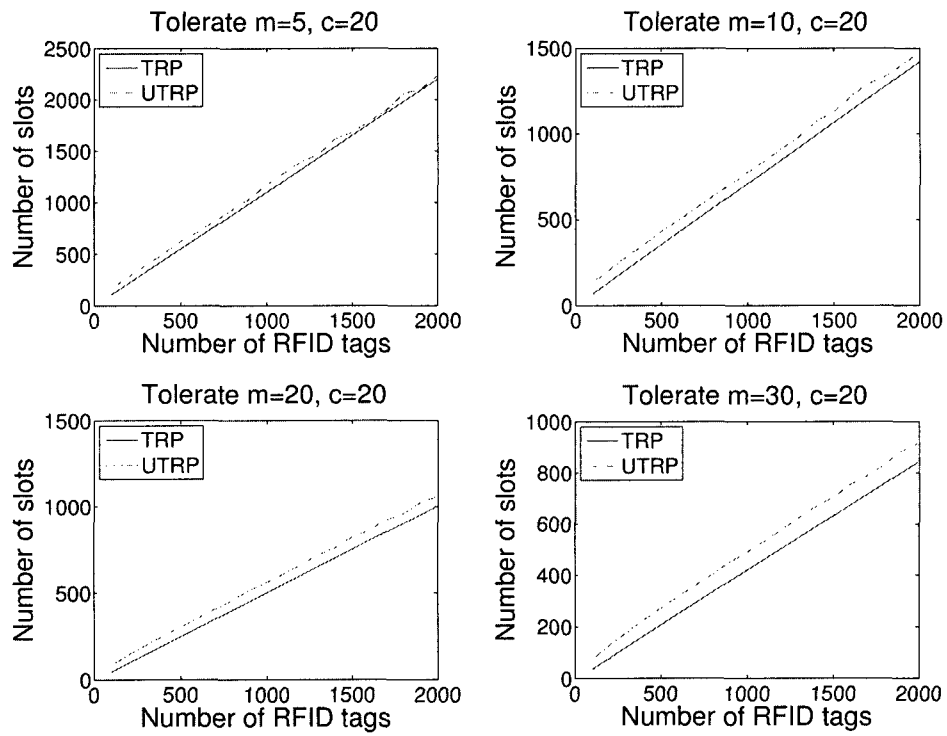


Figure 3.6: Comparing TRP versus UTRP

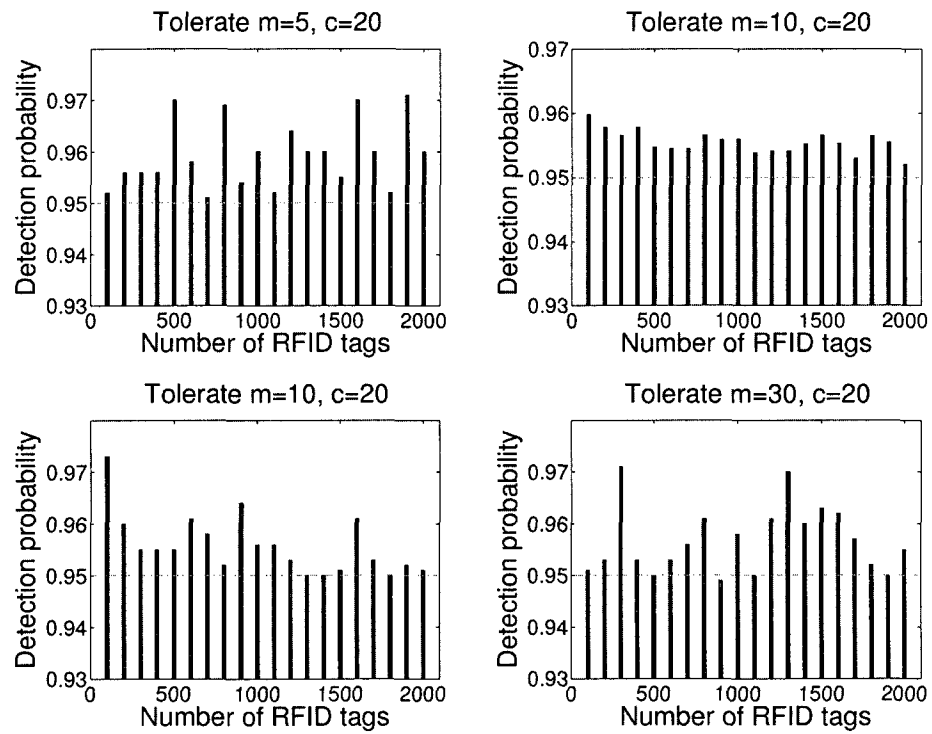


Figure 3.7: Accuracy of UTRP with  $\alpha = 0.95$

## Chapter 4

# Microsearch : A Search Engines for Small Devices

This chapter considers local data extraction for a different type of small device, the sensor. A sensor can store large amounts of information which, given its limited memory and power resources, makes secure and efficient data extraction challenging. This chapter presents Microsearch, our secure search system that modifies information retrieval algorithms to fit within the resource limitations of a sensor.

Pervasive computing allows users to interact with their physical environment just as they would a laptop. A tourist can just as easily interact directly with a signpost for directions as he would a navigate a website. Attendants in a conference can obtain minutes of the previous meeting by querying the conference desk instead of obtaining the data from the group wiki. Such applications all rely on small devices embedded into everyday objects and environment.

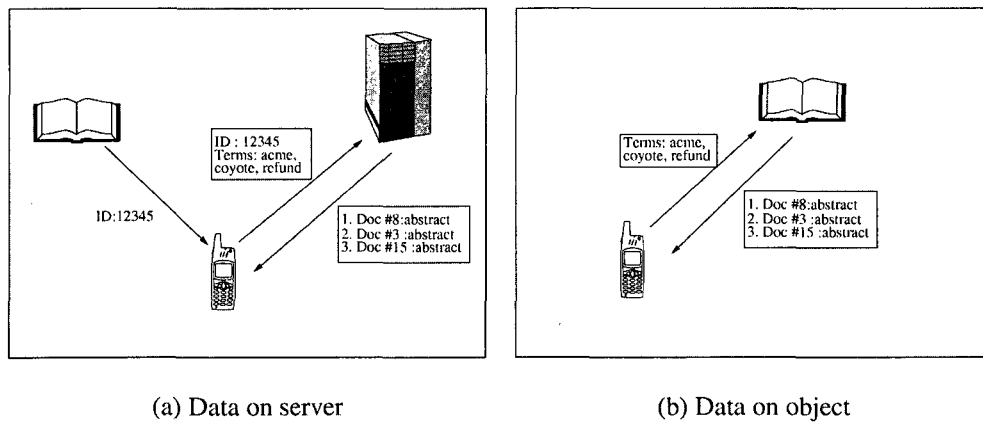
In this chapter, we describe Microsearch, a search system designed for small embedded devices. We use the following example to illustrate how Microsearch can be used. Consider a

collection of document binders. Each binder is embedded with a small device running Microsearch. Each device contains some information about the documents found in that binder. When a user wishes to find some documents, he can query a binder using some terms, i.e. “acme,coyote,refund”, and Microsearch will return a ranked list of documents that might satisfy his query. Also included in the reply is a short abstract of each document to help him make his decision. Later, the user decides to add some notes to a document. Through input devices such as a digital pen [72] or PDA, the user can store notes into each binder. Microsearch will index the user input for future retrieval.

Microsearch is designed to run on resource constrained small devices capable of being embedded into everyday objects. An example of a small device is manufactured by Intel [53] which has a 12MHz CPU, 64KB of RAM, 512KB of flash memory, and wireless capabilities, all packaged in a 3x3 cm circuit board. Larger storage capacity can also be engineered to store more data. In this chapter, we use the terms “mote” and “small device” interchangeably.

Similar to desktop search engines like Google Desktop [47], Spotlight [5] or Beagle [13], Microsearch indexes information stored within a mote, and returns a ranked list of possible answers in response to a user’s query. We envision that Microsearch can be an important component in physical world search engines like Snoogle [115] or MAX [121].

Earlier pervasive systems [2, 27, 28, 91, 101] typically embed simple RF devices like RFID tags onto physical objects. Each tag contains a unique ID identifying that particular object. Data about physical objects are stored remotely on large servers, and are indexed by their respective IDs. To send or receive information regarding a physical object, a user first obtains the ID from the object, and then contacts the remote server with the ID. This design paradigm embeds very simple devices into physical objects, and relies on powerful servers for computation.



**Figure 4.1:** (a) Utilizing backend server. (b) No backend server

As embedded devices become more powerful, a different design paradigm which does not utilize a backend server can be used. Instead of embedding a simple RF beacon into an object, a more powerful device is embedded. Information previously kept on a server will now be stored directly on this device. User queries will also be resolved by the object itself. Fig. 4.1 illustrates the two approaches.

There are several advantages in using a more powerful device to store and retrieve data, rather than relying on a server.

- **Data accessibility.** Storing the data on a more powerful embedded device instead of a remote server allows the user to obtain data directly from the object using protocols like Bluetooth. In areas without long range wireless networks, a simple RF device will only give the user an ID and no other useful information.
- **Simple deployment.** An object embedded with a powerful device can be used without additional configuration, whereas a simple RF device still needs to be configured with a remote server before it can be deployed.

- Intuitive ownership transfer. Storing data directly on the device allows the use of more intuitive security protections for ownership transfer. When user A hands over the physical object to user B, user A no longer can access the stored data since he no longer has access to the object itself. When using a simple RF device, when user A hands over the physical object to user B, the associated data still resides on user A's servers, allowing user A access even though he no longer possesses the physical object.

Despite the advantages, there are two main drawbacks for using a more powerful device. The first is **cost**. Simple RF devices like RFID tags are inexpensive, each tag costing several cents, compared to tens of dollars for a more powerful device. Since RF devices are so much cheaper, several of such devices can be attached to the same object to improve reliability against a damaged tag. The relative higher cost rules out deploying multiple more powerful devices on the same object. The second drawback is **maintenance**. The more powerful device will require periodic maintenance such as replacing the batteries, whereas a simple RFID tag once attached is essentially maintenance free.

Given the strengths and weaknesses, no approach is suitable for all applications. Pervasive applications involving multiple owners and objects operating mostly in an outdoor environment are likely to benefit from using a more powerful device over a weaker RF device. The simple ownership transfer property makes it easier to manage data when the object has to move between different owners. Also, the lack of wireless connectivity makes communication with remote servers unreliable. Examples of such applications include tracking packing crates which may want to record contents and notes as the crate moves from one location to another. By storing the data directly on the crate, the data is available only to the new owners holding the crate.

The challenge of designing Microsearch lies in engineering a complete solution that can run

efficiently on a resource constrained platform. Desktop search systems typically require large amounts of RAM to perform indexing. Similarly, query resolution algorithms usually store intermediate results in memory while resolving a query. With just kilobytes of RAM to spare, it is impossible to port existing solutions directly onto motes. In addition, mote hardware uses flash memory for persistent storage. While conventional flash file systems [32, 119] have been designed, they require more memory than is available on a mote. This necessitates a different system design.

We make the following contributions in this chapter.

- We provide a system architecture that effectively utilizes limited memory resources to store and index different inputs.
- Our architecture incorporates information retrieval (IR) techniques to determine relevant answers to user queries.
- Since conventional IR techniques are designed for more powerful systems in mind, we introduce a space saving algorithm to perform IR calculations with limited amounts of memory. Our algorithm can return the top-k relevant answers in response to a user query.
- A theoretical model of Microsearch is presented to better understand how to choose different system parameters.

## 4.1 Related Work

Desktop search engines are a mainstream feature found in most modern operating systems. In general, these search engines collect metadata from every file, and store the metadata into an inverted index, a typical data structure used to support keyword search [36]. Information retrieval



algorithms [37,40,41,61] are then used to determine the best answer to a query. Our work draws from the basic principals of IR to rank query results.

A counterpart to Microsearch is PicoDBMS [88], a scaled down database for a smart card. PicoDBMS allows data stored inside the smart card to be queried using SQL-like semantics. The main design difference between our work and PicoDBMS is that PicoDBMS uses a database design. Their approach works well in a specific domain like storing health care information, which can enforce structured inputs with specified attribute terms, and assume well trained personal. Microsearch on the other hand uses a search engine design which allows for unstructured inputs without enforcing pre-specified attributes, and a natural language query interface. The relationship between the two can thus be summed up as the differences between a search engine and a database.

We proposed an embedded search system in [115] which allows one to search the physical environment, but focused on integrating a hierarchy of sensors that can cover a large area rather than on how an individual embedded device manages data. Our later work [104] considered the problem of building an information management system on a single sensor. However, [104] does not provide any security solution to protect the data. Furthermore, in this chapter, we improved on the theoretical model found in [104], and evaluated its accuracy.

Low level flash storage systems on the sensor platform have only recently gained interest among researchers. Earlier sensor storage research treated the low level storage as a simple circular log structure. Efficient Log-Structured File System (ELF) [33] was the first paper that introduced a file system especially tailored for sensors, providing common file system primitives like append, delete and rename. Another file system is Transactional Flash File System (TFFS) [43] which deals with NOR flash. Both research are different from ours in that they provide a sensor *file system* and not a sensor *search system*. A search system emphasizes good indexing and query

response time while a file system does not.

Closer to our work is MicroHash [122] which focuses on efficient indexing of numeric data using the sensor flash storage. It creates an index for every type of data monitored by the sensor, for example, temperature or humidity. Since the data indexed by MicroHash is generated by the sensor itself, the index size can be predetermined from the sensor hardware specifications. For example, if the sensor hardware supports temperature monitoring between 10 and 50 degrees at 1 degree granularity, the index with 40 entries can hold all possible data generated. An adaptive algorithm is applied to repartition the index to improve performance. Our research differs from MicroHash in two main ways. First, we allow indexing of arbitrary type of terms, not just numeric ones, and second, we adopt information retrieval algorithms to reply to queries.

Systems like Journaling Flash File System [119], Yet Another Flash File System [32] are designed primarily for larger devices, making them unsuitable for the sensor platform. We refer to [42] for more details. One interesting exception is Capsule [75], which provides object primitives like a stack or index for other sensor applications. These object primitives are designed to work on sensor platform. Unfortunately Capsule's index primitives require the indexable data set to be known before hand, making it unsuitable for indexing the generic metadata. There is also no retrieval algorithm for ranking query results.

File system search is a mainstream feature in most modern operating systems. Since most desktop search systems are commercial offerings, detailed system design is unavailable. However, most search systems share some common functionalities. Metadata for every file is collected and stored in an index. This index data structure in its simplest form resembles an inverted table [40], where given a term, it returns the location of the file containing that term. Information retrieval algorithms [37,40,41,61] are used to determine the best answer to a query.

## 4.2 System Architecture

We begin with describing the inputs to Microsearch. We assume that a user uploads information to Microsearch via a wireless connection through a suitable interface like a PDA. Microsearch requires every user input to consist of two segments, a *payload*, and a *metadata*. The payload is the actual information the user wishes other people to download. The metadata is a description of the payload data, and is used to determine whether a payload is relevant to a user's query. Both the payload and metadata are user generated.

The metadata is essentially a list of terms describing the corresponding payload. Microsearch requires each term, known as a *metadata term*, to be accompanied by a numeric value, known as a *metadata value*, indicating how important *that* term is in describing the payload. A metadata using  $n$  metadata terms to describe a payload can be represented as  $\{(term_1, value_1), \dots, (term_n, value_n)\}$ . For a text based payload, the simplest method to determine the metadata value for a term is to count the number of times that term appears in the payload. Metadata values for non-text based payloads can be defined by the user.

### 4.2.1 Microsearch Design

Microsearch maintains two data structures in RAM: a buffer cache, and an inverted index. The buffer cache is used to temporarily store and organize data before writing to flash to improve overall performance. The inverted index is used to track and retrieve the stored data. In general, when receiving an input file, Microsearch stores the payload into flash memory, and the metadata into the buffer cache. This continues as more inputs are sent to Microsearch until the buffer cache is full. Selected metadata entries are then organized and flushed to flash memory to free up space in the buffer cache, and the inverted index is updated.

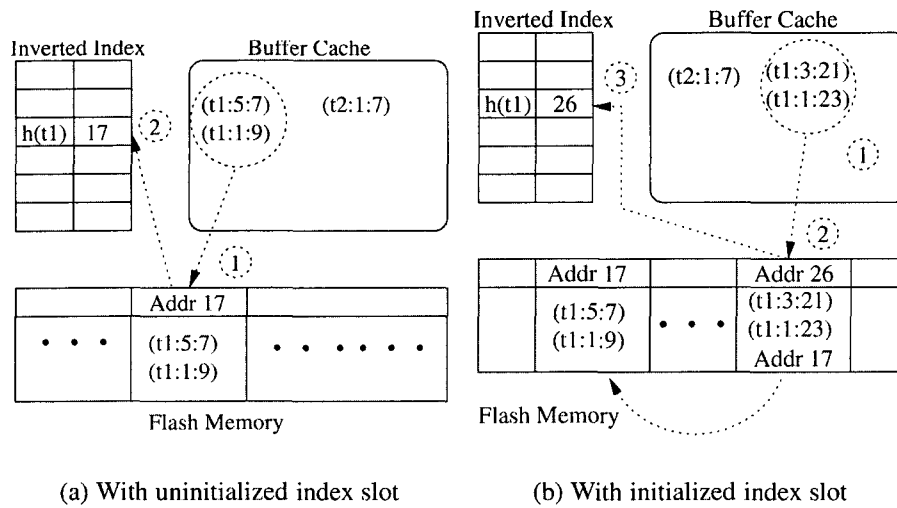
**Receiving an input:** Upon receiving an input file, Microsearch first stores the metadata into RAM, and then writes the payload directly to flash memory. The starting address of the payload in flash is returned and added to each metadata entry for that payload. With this payload address, Microsearch can recover the entire payload if needed. Each metadata entry in the buffer space now becomes a tuple,  $(term, value, address)$ , consisting of a metadata term, a metadata value, and a payload address. For example, consider Microsearch writing a payload to flash memory location  $addr_3$ . All metadata associated with this payload becomes,

$$\{(term_1, 3, addr_3), \dots, (term_n, 2, addr_3)\}.$$

As mentioned earlier, flash memory is used as permanent storage for user inputs. Microsearch writes data to flash memory using a log structure style write which treats the entire flash memory as a circular log, always appending new data to the head of the log. A pointer indicating the next available location in flash memory is kept by Microsearch. Log-style writes have been found to be suitable for flash memory [42]. Since writes are performed on a page granularity, Microsearch will always attempt to buffer the data into at least a single page before writing to flash.

**Buffer cache organization:** As more payloads are sent to the buffer cache, the buffer cache becomes a collection of metadata entries which describe the different input files stored in the mote. There is no longer the concept of a set of entries belonging to a particular payload. Instead, metadata entries which have the same metadata term are grouped together. For instance, two different input files may share some common metadata terms. Inside the buffer cache, the tuples with the same metadata terms are grouped together. For instance, two payloads stored in address  $addr_3$  and  $addr_8$  may share the same term  $term_1$ . Thus, inside the buffer cache, they will be grouped as  $\{(term_1, 2, addr_3), (term_1, 5, addr_8)\}$ .

**Inverted index:** An inverted index is commonly used in search engine systems to retrieve the



**Figure 4.2:** (a) 1) Flushes tuples from buffer cache, 2) Copies address ,  $addr_{17}$ , into inverted index. (b) 1) Copies previous metadata page address from inverted index. 2) Flushes tuples from buffer cache. 3) Copies new address,  $addr_{26}$ , into inverted index.

archived information. A conventional inverted index has every slot correspond to a different term. Each slot stores a pointer to a list of documents or web pages containing that term. By matching a given query term with the inverted index, one can retrieve all the documents or webpages containing that term.

Microsearch uses a modified inverted index which differs from a conventional design in two ways. First, Microsearch uses a hash function to map multiple metadata terms to a certain slot in the inverted index. This results in a smaller inverted index which uses less RAM but is slightly inaccurate. We discuss how Microsearch resolves this inaccuracy in the next section. Second, Microsearch has each slot in the inverted index store the flash address of a page in flash memory containing a group of metadata terms which hash to the same slot. This flash page is known as a *metadata page*. An inverted index slot which already has metadata terms hashed to it is considered *initialized*.

**Buffer eviction with uninitialized index slot:** When the buffer cache reaches full capacity,

tuples will have to be evicted to free up space for new entries. Microsearch hashes the tuples and selects the largest group with the same hashed result, and looks up the corresponding slot on the inverted index. If no metadata term has been hashed to that slot before, that slot is considered uninitialized. Microsearch organizes the group of tuples in the order of their arrival into the buffer cache, and writes the metadata pages into flash memory. If the group of tuples spans multiple flash pages, each metadata page contains the flash memory address of the next page. The address of the *last* metadata page containing the tuples is returned to the inverted index. The inverted index stores this address into the uninitialized slot. The slot is now considered initialized. Fig. 4.2(a) illustrates this process.

**Buffer eviction with initialized index slot:** In the event that an inverted index slot has already been initialized, Microsearch first reads the page indicated by the address found in the index. Microsearch tries to add all the new tuples to that page. If there is not enough space, a new metapage is created and the address from the index will be copied onto the *first* metadata page of tuples. The group of tuples are written to flash memory as before, and the address of the last metadata page is returned and stored in the inverted index. The inverted index thus will always have the address of the latest metadata page written into flash memory. Since each metadata page in flash memory contains the address location of the preceding page, every metadata page can be retrieved by traversing the links. We consider this a *chain* of metadata pages. Fig. 4.2(b) illustrates this process.

**Data deletion:** Deletion in flash memory occurs at a sector granularity. A sector consists of many pages. Each page is 256B and a sector is typically 64KB. A delete pointer is kept by Microsearch to indicate which is the next sector to erase. Once the flash is reaching full capacity, Microsearch frees up storage space by deleting the sector indicated by the pointer. Both payload

pages and metadata pages in that sector are deleted.

Deleting a sector may cause a metadata page to point to a payload page that has already been deleted. We can use the delete pointer to determine what address have already been deleted. Microsearch ignores these payloads when returning data to the user. Deleting a sector may also cause a metadata page to point to another metadata page which no longer exists. Microsearch uses the delete pointer to determine if a metadata page has been deleted, and considers the chain of metadata pages to have terminated.

### 4.3 Secure Microsearch

Security is not a part of conventional search engines, but is an important consideration for embedded search engines. This is because while the servers used to run a conventional search engine can be kept in a secure location, small devices running Microsearch are deployed on physical objects that are easily misplaced or stolen.

#### 4.3.1 Threats

As mentioned earlier, one advantage of storing data on the object instead of a server is simple ownership transfer. Once an object is handed off to a new owner, the previous owner automatically loses access to the stored data since he no longer has physical possession. However, this does not make Microsearch secure against an adversary that is in the vicinity of the user's object.

1. **Privacy attack:** The adversary can query the user's object to obtain some private information.
2. **Storage DoS attack:** The adversary can repeatedly send fake information to the user's object to deplete the storage space.

3. **Query spoofing attack:** The adversary can use a malicious device to fool the user into querying the adversary's device instead of his own, and thus returning incorrect information to the user.
4. **Storage spoofing attack:** The adversary tries to induce the user to store data onto the adversary's malicious device instead of his own. This effectively "deletes" the user's object since no data is ever stored.

#### 4.3.2 Straw Man Protocols

Given the high cost of updating embedded device software once deployed, additional consideration in the design phase will be useful. Here we consider several protocols which appear to provide adequate security but in actuality contain vulnerabilities.

**Blanket encryption:** One apparent solution is for the user to first create a secret key, and encrypt *all* the data before sending it to his object running Microsearch. Since Microsearch's indexing does not distinguish between ciphertext and plain text, no additional modifications are necessary. Since the adversary does not know the user's secret key, querying Microsearch only yields ciphertext which do not reveal the user's private information. This defends against the privacy attack. Blanket encryption also defends against the query spoofing attack. When the user receives any reply, he will decrypt using his secret key. Without knowing the secret key, the adversary cannot generate the correct ciphertext to respond to the user's query. The adversary will be detected when the user cannot decrypt a response.

However, blanket encryption does not defend against a storage DoS attack. The adversary can still store a lot of fake information for Microsearch to index, and thus deplete the storage space. The user is also vulnerable to the storage spoofing attack since he does not know whether his data



has been stored on his own device.

**Simple user only authentication:** Another solution is for the user to first generate a public and private key pair,  $PK$  and  $SK$ . He then stores  $PK$  into his object running Microsearch. When the user wishes to store data or query Microsearch, he executes the following protocol,

$$\text{User} \rightarrow \text{Microsearch} : \text{Request} \quad (4.1)$$

$$\text{User} \leftarrow \text{Microsearch} : PK\{n\} \quad (4.2)$$

$$\text{User} : SK\{PK\{n\}\} = n' \quad (4.3)$$

$$\text{User} \rightarrow \text{Microsearch} : n' \quad (4.4)$$

$$\text{Microsearch} : \text{If } n' = n, \text{ continue,} \quad (4.5)$$

else terminate session

where  $n$  is a random number generated by Microsearch,  $PK\{n\}$  is encrypting  $n$  with the public key  $PK$ , and  $SK\{PK\{n\}\}$  is applying the secret key  $SK$  to a bundle encrypted with  $PK$ . Note that Microsearch will generate a new  $n$  for each new request.

We see in step (2) that Microsearch encrypts a random number  $n$  with the public key. The value of  $n$  is obtained by applying the corresponding  $SK$  which is only known to the user. Thus, only the user can return the correct value of  $n'$  to Microsearch in step (4) which will match the original  $n$ . Without knowing the correct  $n$ , Microsearch will no longer process the user's request.

This protocol defends against the privacy attack and the storage DoS attack. The adversary does not know the secret key  $SK$ , and thus will not be able to return the correct  $n$  in step (4), leading Microsearch to terminate the session. This way, the adversary cannot query data nor add fake data to the object.

However, the simple user only authentication protocol is still vulnerable to attacks 3 and 4.

The adversary will follow the same steps, but in step (5) will not check if  $n' = n$ . Instead, the adversary will always continue to process the user's request. Since the user knows the  $SK$  and expects to be authenticated, he will continue accessing the adversary's device as it were his own.

**Hybrid solution:** An apparent alternative is to combine the two straw man solutions together by running the simple user only authentication protocol *and* encrypting everything. However, this does not defend against the storage spoofing attack since the adversary's device can accept the user's encrypted data.

**Simple mutual authentication:** The problem with the simple user only authentication is that the user never verifies if the object processing his request belongs to him. A straightforward approach seems to be for the user to authenticate the device as well. The user first creates two public and private key pairs, one for himself,  $PK_u$  and  $SK_u$ , and the other for his object running Microsearch,  $PK_o$  and  $SK_o$ . The user stores  $PK_u$  and  $SK_o$  in the object. He then interacts with his object as follows

$$\text{User} \rightarrow \text{Microsearch} : \text{Request}, PK_o\{n\} \quad (4.1)$$

$$\text{Microsearch} : SK_o\{PK_o\{n\}\} = n' \quad (4.2)$$

$$\text{User} \leftarrow \text{Microsearch} : PK_u\{n'\} \quad (4.3)$$

$$\text{User} : SK_u\{PK_u\{n'\}\} = n'' \quad (4.4)$$

$$\text{User} : \text{If } n'' = n \quad (4.5)$$

$$\text{User} \rightarrow \text{Microsearch} : n'' \quad (4.6)$$

$$\text{User} : \text{Else terminate session.} \quad (4.7)$$

$$\text{Microsearch} : \text{If } n'' = n', \text{ process request} \quad (4.8)$$

else terminate session.

This protocol appears to allow both the user and his object to authenticate each other. If the object belongs to the adversary, it will not know  $SK_o$ , and cannot obtain  $n'$  which is equal to  $n$ . As a result, when the user decrypts the adversary's reply in step (4), the user will observe  $n'' \neq n$ , and conclude that the object does not belong to him. Now considering the adversary trying to query the user's object. Since the adversary does not know  $SK_u$ , the adversary cannot send the correct  $n''$  such that  $n'' = n'$ , resulting in the user's object declining to process the request. This protocol improves on the simple user only authentication, and appears to defend against all attacks listed above.

However, an adversary can launch an effective privacy and storage DoS attack by ignoring the object's reply in step (5). Since the adversary selects the initial random number  $n$ , he can always return the same value in step (6). The object will then verify that  $n'' = n'$ , and process the request.

### 4.3.3 Single User Protocol

The problem with the simple mutual authentication is the repeated use of the same random number. To be secure, two random numbers must be used, one generated by the user, and the other by the object. Assuming that we have only one user, we use the same setup as the simple mutual authentication, but executes the following protocol.

$$\text{User} \rightarrow \text{Microsearch} : \text{Request}, PK_o\{n_1\} \quad (4.1)$$

$$\text{Microsearch} : SK_o\{PK_o\{n_1\}\} = n'_1 \quad (4.2)$$

$$\text{User} \leftarrow \text{Microsearch} : n'_1, PK_u\{n_2\} \quad (4.3)$$

$$\text{User} : \text{If } n'_1 = n_1 \quad (4.4)$$

$$\text{User} : SK_u\{PK_u\{n_2\}\} = n'_2 \quad (4.5)$$

$$\text{User} \rightarrow \text{Microsearch} : n'_2 \quad (4.6)$$

$$\text{User} : \text{Else terminate session} \quad (4.7)$$

$$\text{If } n'_2 = n_2, \text{ continues} \quad (4.8)$$

else terminate session

where  $n_1$  is a random number chosen by the user, and  $n_2$  is the random number chosen by the object.

When the user sends  $PK_o\{n_1\}$  to the object, only his object knows  $SK_o$  to decrypt and return the correct  $n_1$ . Thus at step (4) if  $n'_1 = n_1$ , the user knows that the object belongs to him. Similarly, the object authenticates the user by sending  $PK_u\{n_2\}$  which can only be decrypted by  $SK_u$  which is only known to the user. Thus in step (8), only the user can return the correct  $n'_2 = n_2$ , at which point the object can trust that the request was not issued by an adversary.

We see that this protocol defends against both the privacy attack and storage DoS attack since the adversary cannot return the correct  $n'_2$ , causing the object to terminate the session. Both query spoofing attack and storage spoofing attack are also foiled since the adversary's device cannot return the correct  $n'_1$  to the user, causing the user to terminate the session.

#### 4.3.4 Multiple users

We can extend the single user protocol above to accommodate  $k$  users by storing  $PK_1, \dots, PK_k$  in the object. The user that sends a request will indicate which public key to use, and the rest of the protocol can be executed. However, the problem is that storing a large number of keys will take up limited storage space in the embedded device.

We assume that the owner of the object creates a master public and private key pair,  $PK_{mas}$  and  $SK_{mas}$ , as well as a key pair for the object,  $PK_o$  and  $SK_o$ . He stores  $SK_o$  and  $PK_{mas}$  into the object. Each valid user,  $u$ , will be issued his own public and private keys,  $PK_u$  and  $SK_u$ , and a certificate  $cert$  where

$$cert = SK_{mas}\{PK_u\}$$

Now, when the user wants to access the object, he will use the single user protocol, but include his  $cert$  and public key  $PK_u$  in the request. The object can apply the master public key to verify that  $PK_u$  is valid,

$$PK_{mas}\{cert\} = PK_{mas}\{SK_{mas}\{PK_u\}\} = PK_u.$$

Since only the object owner knows  $SK_{mas}$ , verifying  $PK_u$  through  $PK_{mas}\{cert\}$  indicates that this public key is authorized by the owner. The rest of the interaction remains the same as the single user protocol.

#### 4.3.5 Performance Details

The challenge of implementing security on an embedded device is hardware limitations. With limited computation and battery power, conventional security protocols cannot be directly applied. The protocols given earlier rely on public key cryptography, which can be implemented using different cryptographic primitives such as RSA, ElGamah, or Elliptic Curve Cryptography (ECC).

A security protocol for Microsearch will inevitably be application and deployment specific. Since different types of users will have different requirements, we show in Table 4.1 several important parameters such as the size of a public key, and the time needed to encrypt some data. The values are derived from an embedded device with 8MHz processor and 10KB RAM. The table can serve as a guideline for designers in estimating the cost of their protocols.

Parameter	RSA	ECC
Secret key size	128 B	20 B
Public key size	128 B	40 B
Certificate size	128 B	40 B
Ciphertext overhead	128 B	60 B
Encryption time	21 s	2.8 s
Decryption time	21 s	1.4 s

**Table 4.1:** Parameters for RSA and ECC encryption.

The ciphertext overhead in Table 4.1 is the additional number of bytes that result from encrypting the data. In other words, encrypting a 16 byte data using RSA will result in a 144 byte ciphertext. Furthermore, in practical security implementations, we typically do not use public keys to encrypt data. Instead, we use a symmetric key to encrypt the data and then use the public key to encrypt the symmetric key. A typically symmetric key is 16 bytes in size.

## 4.4 Query Resolution

Query resolution describes the process of returning an accurate answer to a user's query. A user queries a mote by sending a list of search terms and parameter  $k$  which specifies the top- $k$  rankings

he is interested in. The user receives an ordered list of  $k$  possible payload data as an answer. We begin by first introducing a basic query resolution algorithm. The actual space saving algorithm used by Microsearch is presented later.

#### 4.4.1 Information Retrieval Basics

Microsearch uses a simple information retrieval weighing calculation, the TF/IDF function, to determine how relevant each payload address is in satisfying the user's query. Under the TF/IDF function, the weight of each metadata term of a payload is determined by the product of  $TF \cdot IDF$ , where TF is the metadata value of the metadata term, and IDF is  $\log(\frac{N}{DF})$ , where  $N$  is the total number of payloads stored within the mote, and  $DF$  is the number of payloads which share the same metadata term. The relevancy of a payload, or the score of the payload, is the combined weights of the metadata terms matching the search terms.

For example, let Microsearch contain a total of 5 payloads. One of the payload  $p1$  have tuples  $(term_1, 3), (term_2, 2)$ , and another payload  $p2$  have a tuple  $(term_1, 6)$ . The remaining payloads do not contain terms  $term_1, term_2$ . A user issues a query  $(term_1, term_2)$ . Clearly, the ideal answer should be  $p1$  since it is the only payload that contains both terms. Using TF/IDF, the weight of  $p1$  will be

$$3 \cdot \log \frac{5}{2} + 2 \cdot \log \frac{5}{1} = 5.96,$$

and weight for  $p2$  will be

$$6 \cdot \log \frac{5}{2} = 5.49.$$

From the calculation, we see that  $p1$  has a larger final score than  $p2$  despite  $p2$  having a larger term score 6.

#### 4.4.2 Basic Algorithm

In the basic algorithm, Microsearch first obtains a set of metadata entries with metadata terms which match the search terms. Remember that a metadata entry is of the form  $(term, value, address)$ . With this chosen set of metadata entries, Microsearch then ranks the payload addresses in order of their relevancy, and uses the top ranking addresses to retrieve the payloads to return to the user. Since each payload has a unique flash memory address, this address is used as an identifier for a payload.

To obtain the set of metadata entries, Microsearch first scans all the metadata entries in the buffer cache for metadata terms matching the search terms. Matching entries are then copied to a separated section of RAM. Next, Microsearch uses the inverted index to find matching metadata entries in flash. Microsearch first applies the hash function to each search term to determine the corresponding slot in the inverted index. These slots contain the addresses of the metadata pages in flash memory. Each metadata page contains metadata terms which hash to the same slot. Note that the metadata terms found in the same page do not necessarily have to be the same. They only need to hash to the same slot. Microsearch then retrieves each metadata page one at a time until all metadata pages are read. For each metadata page read, Microsearch compares the actual metadata terms to the search terms, and copies the matching ones to RAM.

At this point, Microsearch has a list of all metadata entries which match the search terms. Microsearch uses the TF/IDF function to determine the score for each payload address, and orders them from the highest score to the lowest. Microsearch then uses the top  $k$  payload addresses to obtain the actual payloads from flash to return to the user.



### 4.4.3 Performance Improvements

The basic algorithm first selects all the metadata entries which match the search terms, and then proceeds to eliminate low scoring payload address. This approach requires a large section of RAM to be set aside. A better solution is to eliminate low scoring payload addresses as they are encountered.

There are two difficulties in deriving a better solution. First, Microsearch relies on TF/IDF calculations to determine the relevancy of each payload address. Calculating the IDF requires knowledge of DF, the number of payloads in flash which share the same metadata term. The DF score can only be obtained by reading in every metadata page from flash and checking the corresponding metadata terms. We cannot maintain a DF score for each inverted index slot since there could be multiple metadata terms that hash to the same slot.

Second, even if we use only TF score without IDF, a simple elimination scheme does not work. Consider the example when a user queries Microsearch with two search terms  $x$  and  $y$ , with  $k = 1$ . For simplicity, we assume that the buffer cache is empty, and  $x, y$  hash to different slots in the inverted index, i.e.  $hash(x) \neq hash(y)$ . We have 10 metadata pages each in flash memory matching  $hash(x)$  and  $hash(y)$ . Now after reading in the first metadata page for  $x$ , we obtain 2 metadata entries with  $x$ . This means there are two potential payload addresses which can satisfy the user's query. Let us denote these two addresses as  $addr_1$  and  $addr_2$ . The first metadata page for  $y$  does not contain either  $addr_1$  or  $addr_2$ . At this point, even though the user specifies the top-1 answer, we cannot eliminate  $addr_1$  or  $addr_2$  because we cannot determine whether either payload address actually contains the term  $y$ . The reason is that Microsearch does not guarantee that metadata from the same payload are evicted from the buffer cache at the same time. To be sure with  $addr_1$  or  $addr_2$  contain  $y$ , we have to continue reading in the metadata pages for  $hash(y)$

from flash.

#### 4.4.4 Space Efficient Algorithm

To derive a space efficient algorithm, Microsearch exploits the sequential write behavior of log file system. This sequential behavior ensures that data is always written to the flash memory in a forward order. This means that if payload  $p1$  is sent to the mote before payload  $p2$ , then the flash address of  $p1$  will be smaller than that of  $p2$ .

To describe the space efficient algorithm, we first define some notations. We let  $t$  be the number of search terms and a user query is  $\{k, \{st_1, st_2, \dots, st_t\}\}$ . We denote the inverted index as *InvIndex*, and the latest metadata page written to flash memory as the head metadata page. For example,  $InvIndex[hash(st_i)]$  returns the address of the head metadata page for  $st_i$ . We represent this value as  $head[i]$ .

We allocate a memory space  $page[i]$  for each query term  $st_i$ , which is sufficient to load one metadata page from flash memory. We first check the buffer and load the metadata entries whose metadata term is  $st_i$  to  $page[i]$ . If  $st_i$  is not found in the buffer, we load  $head[i]$  to  $page[i]$ . Let  $\min(page[i])$  and  $\max(page[i])$  denote the smallest and largest payload addresses in  $page[i]$  respectively. We define a *cutoff* value as the maximum value among  $\min(page[i])$ , i.e.

$$cutoff = \max\{ \min(page[i]), \forall i \in [1, t] \}.$$

Due to the following lemma 4.1, we have all necessary information to calculate the IR scores for the loaded index entries, whose payload address is greater than or equal to *cutoff*. The entire algorithm is found in Algorithm 8.

**Lemma 4.1** *For any index entry whose payload address  $\geq$  cutoff, if its term field is included in the query terms, it must have been loaded into memory.*

**Proof:** It can be proved by contradiction. Assume there exists such an index entry whose term is one of the search terms  $st_i$ , and payload address is  $p \geq \text{cutoff}$ . In addition, the metadata page it belongs to has not been loaded yet. It means that the contents in  $\text{page}[i]$  are from some preceding metadata page in the same chain, i.e., the loaded metadata page is closer to the chain head. For example, in Fig. 4.2(b), the metadata page at  $\text{addr}_{26}$  precedes the page at  $\text{addr}_{17}$  in the same chain. According to our protocol, if metadata page  $a$  precedes metadata page  $b$ , page  $b$  must be flushed into flash earlier than page  $a$ . Thus, any payload address in  $a$  must be larger than any payload address in  $b$ . Based on our hypothesis, therefore, we can derive that any payload address in  $\text{page}[i]$  must be larger than  $p$ , thus

$$\min(\text{page}[i]) > p \geq \text{cutoff}.$$

It is a contradiction with the definition of *cutoff*, which implies  $\forall i \in [1, t], \min(\text{page}[i]) \leq \text{cutoff}$ .

■

A  $k$ -length array  $\text{result}[k]$  is used to store the intermediate results which are the candidates of final reply. Every time we get a new IR score, this array will be updated to keep the current top- $k$  results. The processed index entries will be eliminated from memory. When  $\text{page}[i]$  is empty, we load the next metadata page in the chain from flash memory and repeat this process. Based on the definition of *cutoff*, there must be at least one  $\text{page}[i]$  becoming empty after each iteration. The algorithm terminates when  $\forall i, \text{page}[i] = \emptyset$  and every chain reaches its tail. In this design, instead of loading every metadata page, we load at most one page for each query term. Thus, the memory space needed is at most  $O(E \cdot t)$ , where  $E$  is the size of a metadata page.

Note that in practice we traverse each index chain twice, the first time to obtain the  $DF$  for the term, and the second time to execute the actual query algorithm. This is done to match the  $DF$  definition in the simple  $TF/IDF$  scoring algorithm adopted for this chapter. If alternative scoring algorithms that do not require this form of  $IDF$  calculation are used, this extra traversal can be avoided.

## 4.5 Theoretical Model

A key parameter in designing Microsearch is the size of the inverted index. We first present the intuition behind the choice of inverted index size, followed by the theoretical model.

With a smaller inverted index, uploading information into Microsearch is faster. When the buffer cache is full, Microsearch evicts data from the buffer cache into flash memory. Microsearch groups all the metadata terms which hash to the same inverted index slot together for eviction. Recall that writing data to flash memory occurs on a page granularity. In other words, the cost of writing a page into flash memory is the same even in situations where there are not enough metadata terms hashing to the same inverted slot to make up a flash page. A smaller inverted index results in more metadata terms hashing to the same inverted index slot. This increases the probability of more entries being flushed out of the buffer cache each time.

With a larger inverted index, query performance may be improved because each index slot will have fewer metadata terms hashing to it. As a result, the chain of metadata pages in flash memory which map to each inverted index slot becomes shorter. When replying to a query, Microsearch has to read in the entire chain of metadata pages. A shorter chain of metadata pages means that fewer pages are needed to be read from flash memory, and thus speeding up query performance

Next, we first analyze a important parameter, the number of metadata terms in each metadata

page. Then we derive query performance and insert performance of Microsearch. Table 4.2 lists some variables in our model.

# of documents	$D$
# of metadata per document	$m$
# of query terms	$t$
Size of main index (# of slots)	$H$
Size of metadata page (# of metadata terms)	$E$
# of actual metadata terms per metadata page	$E'$
Size of buffer (# of metadata terms)	$B$
# of terms flushed per eviction	$x$

**Table 4.2:** System Model Variables

**Analysis of  $E'$  and  $x$ :** Although each metadata page can hold  $E$  metadata terms, the actual number of terms in each metadata page ( $E'$ ) might be less than  $E$ . It depends on the number of metadata terms Microsearch flushes to the flash when the buffer is full. Recall our eviction process described in Section 4.2, the largest group of tuples that hash to the same index slot will be evicted to the flash. Let  $x$  denote the number of metadata terms in this largest group, i.e., every eviction could put  $x$  terms to the flash. According to our protocol, if  $x > E$ , Microsearch will write multiple metadata pages in the flash, where the last page contains  $(x \bmod E)$  terms and the other pages are full with  $E$  terms. Thus, in this case,  $E' = \lceil \frac{x}{E} \rceil$ . Otherwise, if  $x \leq E$ , Microsearch will write at most one new metadata page each eviction. Recall our eviction process will first attempt to pad the last

written metadata page. As a result, every metadata page will contain  $\lfloor \frac{E}{x} \rfloor \cdot x$  terms. In summary,

$$E' = \begin{cases} x / \lceil \frac{x}{E} \rceil & \text{if } x > E; \\ \lfloor \frac{E}{x} \rfloor \cdot x & \text{if } x \leq E. \end{cases}$$

Next, we give an analysis of deriving the value of  $x$ . Based on its definition,  $x$  is obviously at least  $\lceil \frac{B}{H} \rceil$ . For one hashed value  $h_i$ , the probability that  $p$  entries in the buffer map to  $h_i$  is

$$\binom{B}{p} \left( \frac{1}{H} \right)^p \left( 1 - \frac{1}{H} \right)^{(B-p)}.$$

Thus, the probability that at least  $p$  entries map to  $h_i$  is

$$q = \sum_{j \geq p} \binom{B}{j} \left( \frac{1}{H} \right)^j \left( 1 - \frac{1}{H} \right)^{(B-j)}.$$

The probability that  $x \geq p$  is  $P(x \geq p) = 1 - (1 - q)^H$ . Thus,

$$P(x = p) = P(x \geq p) - P(x \geq p + 1).$$

Therefore, the expected value of  $x$  is

$$x = \sum_{i \geq \lceil \frac{B}{H} \rceil}^B P(x = i) \cdot i.$$

The following Fig. 4.3 illustrates an example of  $x$  and  $E'$ .

**Query Performance:** Assume there are  $D$  number of files stored in the flash memory and each of them is described by  $m$  terms on average. Totally, we need store  $D \cdot m$  index entries in the flash, which occupy  $\frac{D \cdot m}{E'}$  metadata pages. Considering a fair hashing, the average length of metadata page chain is  $\frac{D \cdot m}{E' \cdot H}$ . When Microsearch processes a query for one term, based on the hash value of

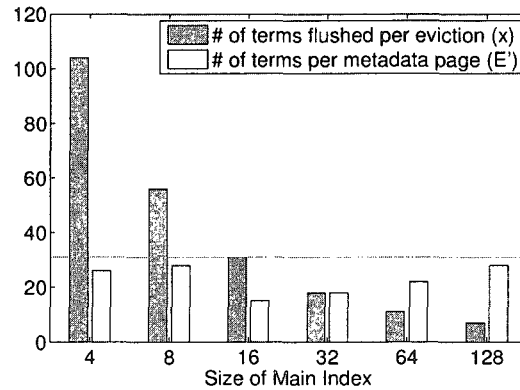


Figure 4.3: Value of  $x$  and  $E'$  v.s. Index Size ( $H$ ): Buffer cache is 5K bytes, buffer size  $B$  is 640. The flat line illustrates the value of  $E = 31$ .

the term, it has to go through one of the metadata page chain twice. One round for collecting the value of document frequency and the other for finding the top- $k$  answers. Expectedly, Microsearch will need to read  $\frac{2 \cdot D \cdot m}{E' \cdot H}$  metadata pages from the flash. For a query for  $t$  terms, Microsearch has to access  $t$  distinct metadata page chains, when  $t \ll H$ . Thus, it takes at most  $\frac{2 \cdot t \cdot D \cdot m}{E' \cdot H}$  page reads to reply. The following Fig. 4.4 illustrates an example of query performance.

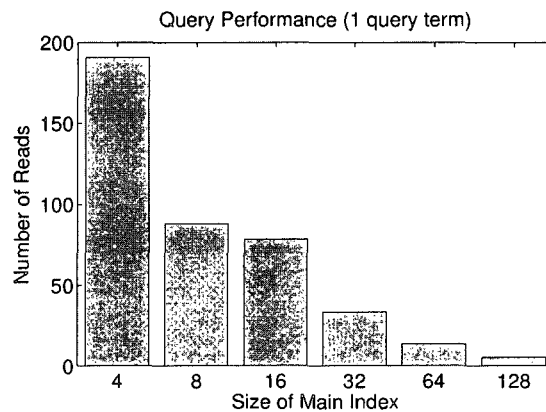
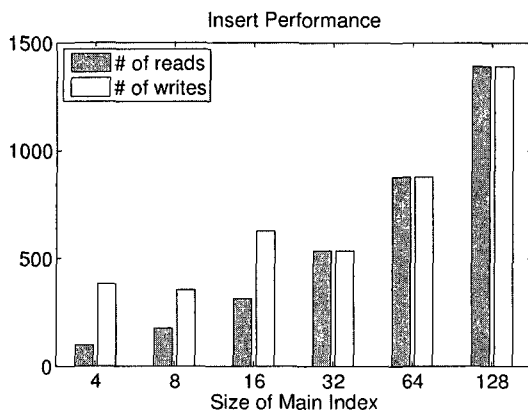


Figure 4.4: Query Performance v.s. Index Size ( $H$ ):  $D = 1000, m = 10, t = 1, E = 31, B = 640$  (5K-byte buffer).

**Insert Performance:** Insert performance is measured by the number of reads and writes operated during inserting  $D$  files. Microsearch only reads once in each buffer eviction. As we

mentioned, each eviction flushes  $x$  metadata terms. Since in total there are  $D \cdot m$  metadata terms, the number of reads for insertion is  $\frac{D \cdot m}{x}$ . The number of writes must be no less than the number of reads. If  $x \leq E$ , each eviction only write on metadata page, thus the number of writes is the same as the number of reads. If  $x > E$ , however, Microsearch will write multiple pages in each eviction process. Since each metadata page contains  $E'$  terms, the number of writes is  $\lceil \frac{D \cdot m}{E'} \rceil$ . The following Fig. 4.5 illustrates an example of query performance.



**Figure 4.5:** Insert Performance v.s. Index Size ( $H$ ): We set  $D = 1000, m = 10, E = 31, B = 640$  (5K-byte buffer).

## 4.6 System Evaluation

### 4.6.1 Hardware and Implementation

We used the TelosB mote for our experiments. TelosB features a 8MHz processor, 10KB RAM, 48KB ROM and 1MB of flash memory. An IEEE 802.15.4 standard radio is used for wireless communication. The entire package is slightly larger, measuring 65x31x6 mm, and weighs 23 grams without the battery. We implement Microsearch using NesC in TinyOS environment and a user interface using Java. The core program takes around 1700 lines of NesC codes and the



interface takes around 800 lines of Java codes.

From Table 4.2, the parameters related to implementation include the size of main index ( $H$ ), the number of metadata terms in each metadata page ( $E$ ), and the size of buffer cache ( $B$ ). In our implementation, we set  $H = 32$ ,  $E = 31$ , and  $B = 372$ . Each entry in  $H$  is three bytes. Each metadata entry,  $(term, value, address)$ , is eight bytes. We use the first five bytes to store the term, three bits to store the value field, and the rest for the address field. The total amount of space allocated to the buffer and index is 3K memory.

#### 4.6.2 Generating Workload Data

A difficulty in evaluating a search system lies in determining an appropriate workload. An ideal workload should consist of traces derived from real world applications. However, since Microsearch-like applications do not yet exist, we cannot collect such traces for evaluation. This also makes it difficult to generate synthetic traces that approximate user behavior. We generated our workload by observing related real world applications.

We envision that most objects such as a wedding photograph album or a document binder will embed a mote running Microsearch. Since each object has its own mote, each mote does not necessarily have to contain a large amount of unique data. For instance, a large bookshelf may contain hundreds of document binders, with a combined total of thousands of documents. However, each binder may contain only a dozen documents. Since each binder embeds a mote, each mote only needs to index the contents of its own binder. Consequently, none of our workload considers excessive large number of unique pieces of data. Our evaluation considers the following two workloads.

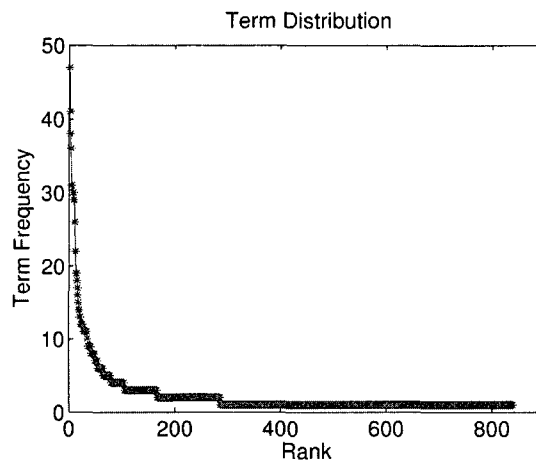
**Annotation workload:** This workload represents a user storing many short pieces of infor-

mation, similar to Post-it reminders or memos, onto a mote. The metadata a user would associate with these type of applications is usually very short. We want a real world application where many users provided annotations, since this closely resembles the metadata we desire. One such application is the annotation of online photographs. We extracted 622 photographs and their accompanying annotations from the website `www.pbase.com`. This created a set of 2059 metadata terms, an average of 3.3 metadata terms per photograph. We consider each photograph as a unique input, and each photograph's annotation as the corresponding metadata terms. The metadata value of each term is set to 1. Fig. 4.6 shows the metadata term distribution for this workload. Recall in Table 4.2, the parameters related to workload are  $D = 622$  and  $m = 3.3$ .

**Doc workload:** This workload represents a mote used for tracking purposes, such as keeping track of the documents inside a binder. We assume that the binder contains academic publications, and the accompanying mote contained the abstracts of all the papers. A user can query Microsearch just like querying *Google Scholar* to determine if a particular paper is inside the binder. To create the doc workload, we extracted 21 papers from the conference proceedings of Sensys 2005, and derived an average of 50 metadata terms for each paper. The metadata terms include author names, paper title, keywords. Metadata values are based on the number of times each term appeared in the paper abstract. Recall in Table 4.2, the parameters related to workload are  $D = 21$  and  $m = 50$ .

### 4.6.3 System Performance

We use the annotation workload to evaluate system performance. The objective is to determine the performance of the two main Microsearch components: indexing the data sent by a user, and replying a user query. Time is the main metric used. In addition, for every evaluation, we



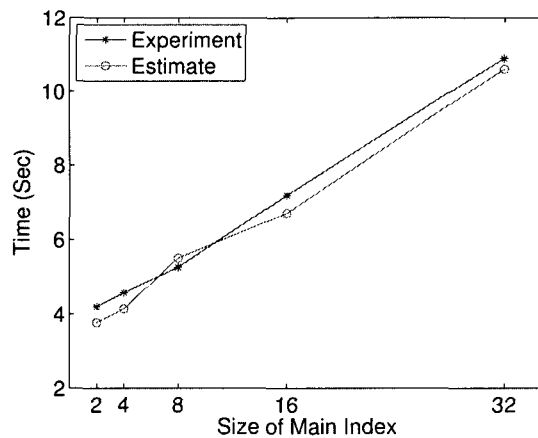
**Figure 4.6:** Term distribution for annotation workload

present both the actual measured performance, and the predicted performance derived from our theoretical model introduced earlier. The closer the predicted results match the actual results, the more accurate our theoretical model is.

To prepare, we first generate a set of queries by randomly choosing terms from the 2059 harvested annotations. We then divided the set of queries into four groups, with the first group containing queries with one search term, the second group with queries containing two search terms and so on. Each group has a total of 100 queries. We limit the number of search terms to at most four terms, since studies conducted on mobile search conclude that most searches consists of between 2 and 3 terms [8, 29, 59].

We then inserted the 622 metadata files with a total of 2059 metadata terms into Microsearch. This is equivalent to inserting 622 short messages into the mote. Fig. 4.7 shows the time taken to insert all the terms into Microsearch. We see uploading information is faster given a smaller inverted index. This is consistent with the intuition given in the prior section.

In Fig 4.8, we show the time taken for Microsearch to answer a user's query. As discussed in the theoretical model, we see that a larger inverted index processes queries faster than a smaller



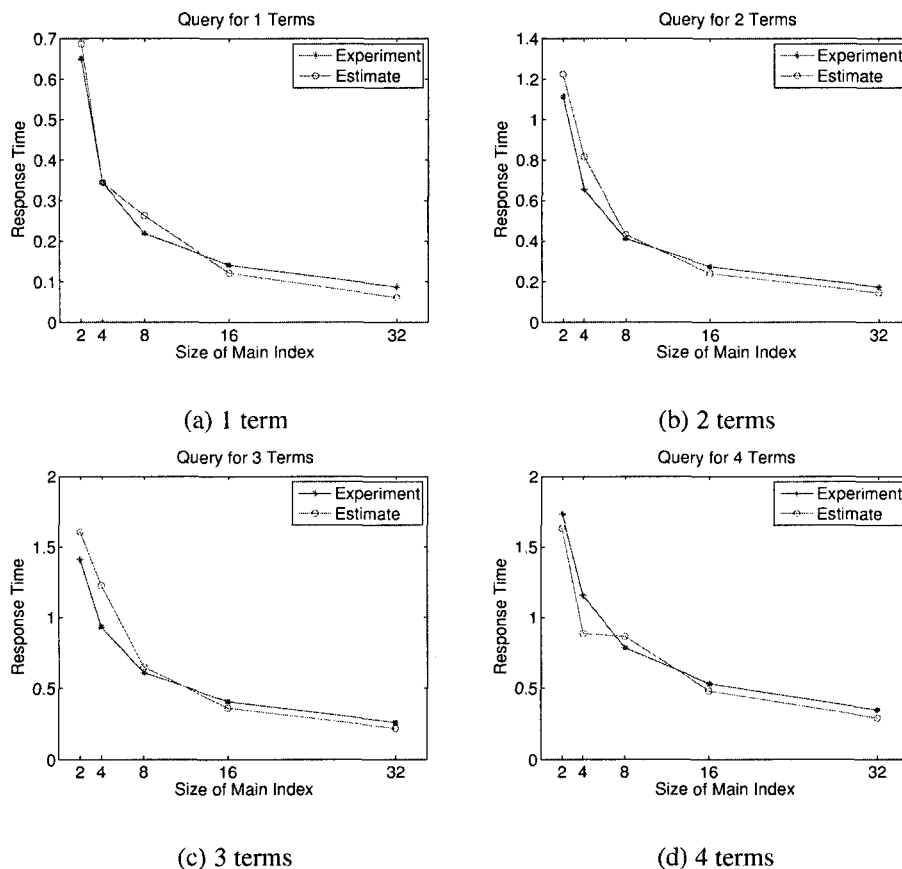
**Figure 4.7:** Predicted and actual insert performance

inverted index. The predicted query response time is also very close to the measured time. Overall, Microsearch is able to answer a user's query in less than two seconds. Fig 4.9 shows the actual overhead of Microsearch minus the time taken to read from flash memory. We see that the additional time taken to rank the query answers is less than 0.5 seconds.

#### 4.6.4 Search Accuracy

The precision verses recall metric is commonly used to evaluate search systems. Precision is defined as the number of relevant items retrieved divided by the total number of retrieved items. Recall is the number of relevant items retrieved divided by the total number of relevant items in the collection. A better search system will consistently returns a higher precision for any given recall rate. However, the precision verses recall metric does not measure how well the search system *ranks* the results.

Shah and Croft [96] suggested using metrics from question answering (QA) research [110] to evaluate search algorithms for bandwidth or power constrained devices. QA is a branch of information retrieval that returns answers instead of relevant documents in response to a query. In

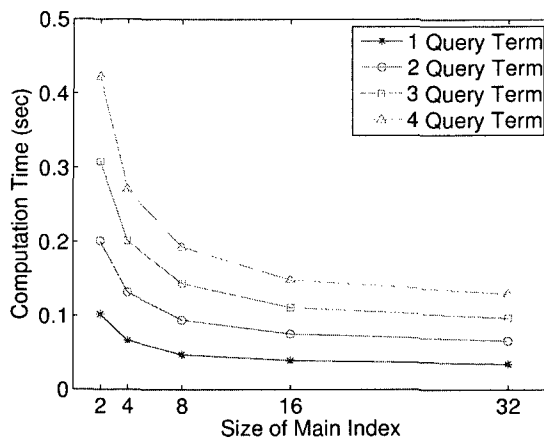


**Figure 4.8:** Predicted and actual query response time measured in seconds

QA research, the goal is to return a single or a very small group of answers in response to a query, not all relevant documents. The main evaluation in QA is the mean reciprocal rank (MRR). MRR is calculated as

$$MRR = \frac{1}{\text{rank of first correct response}}$$

The first correct response is the top ranked document in the model answer. For example, if the model answer to a query is the ranked list  $(A, B, C)$  and IR system returns the list of  $(C, B, A)$ . The first correct answer should be  $A$  and the returned answer is 2 spots off. The MRR for this question is thus  $\frac{1}{3} = 0.33$ . We evaluate the performance of our search system by modifying the guidelines



**Figure 4.9:** Processing time overhead of search system processing

for QA track at TREQ-10 [26]. We consider only the top 3 answers in calculating MRR. If the model answer does not appear within the top three ranks, it has a score of 0.

We use the doc workload to evaluate the accuracy of Microsearch. We first determine a set of queries based on the 21 publications, and their corresponding answers by hand. These questions are divided into three groups, *LastName*, *Title* and *KeyTerms*. The queries for the first two categories are terms from the last names and paper titles of the conference proceedings. The queries for the last category are a mixture of terms from last names, titles and abstract keywords.

Fig. 4.10 shows the results of our search system for the three categories. For each category, we plot the MRR for the different categories over the average of 21 questions. From the figure, our system returns a MRR of 0.95 for both *LastName* and *KeyTerms*. The MRR for *Title* is lower at 0.83, because some of the paper titles contained very common words like “Packet Combining In Sensor Networks”. In all cases, we see that on average Microsearch will return the correct answer when the user specifies  $k = 3$ .

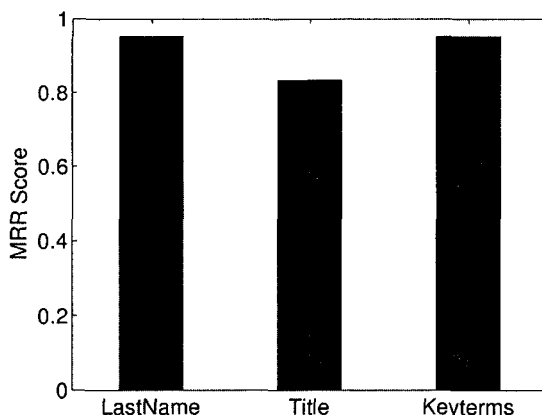


Figure 4.10: Query accuracy (k=3)

#### 4.6.5 Experiment Limitations

We stress that the results shown in Fig. 4.10 do not suggest that Microsearch will yield similar accuracy results in a real deployment. The reason for the very high accuracy results is that we have deliberately avoid using vague queries since it is difficult to objectively quantify what the answer *should* be. Instead, we first generated a set of queries which contain terms that are found in multiple documents, and then manually determine what the correct answers to those queries. In all instances, the answers we select are unambiguous. For instance, given a query “underwater sensor storage”. There is only one paper containing the term “underwater”, and three papers containing the term “storage”. Almost all papers contained the word “sensor”. The correct answer is should be the only paper on underwater sensors despite the two other papers containing more occurrences of the term “storage”.

Microsearch uses TF/IDF calculation to resolve queries, a conventional weighing algorithm widely used in information retrieval research. Our contributions are the space saving algorithm (Algorithm 1) which uses less memory space to compute TF/IDF, and that the results in Fig. 4.10

can only be interpreted as showing the correctness of applying our space saving algorithm technique in determining TF/IDF, and not the accuracy of Microsearch.

#### **4.6.6 Model Accuracy**

As we expect higher level applications to be built above our low level search system, the predictability of our system is an important performance metric. Since sensors, as embedded devices, have more stringent resource limitations, it is important when developing applications to be able to accurately budget the sensor resource. A low level component that gives unpredictable performance will adversely impact sensor application design and deployment. To evaluate our performance, we give a set of requirements and derive the expected performance based on our model. Then, we modify our prototype based on these requirements and compare the experimental results against the expected results. The closer the match, the better the predictability of our system. In all cases, we limit the available RAM for main index and buffer size to *3KB*. Table 4.3 shows our requirements and recommendations. For a given targeted query response time and expected query term length, the model provides a recommendation of size of index. The buffer cache size is then *3KB* subtracted from the index size. An *x* indicates our search system cannot meet the targeted query response time given the hardware requirements. Figure 4.11 shows the results. We see that the experimental results are slightly higher than the targeted query response time when the number of query terms is expected to be larger. The difference is less than 0.05 seconds.

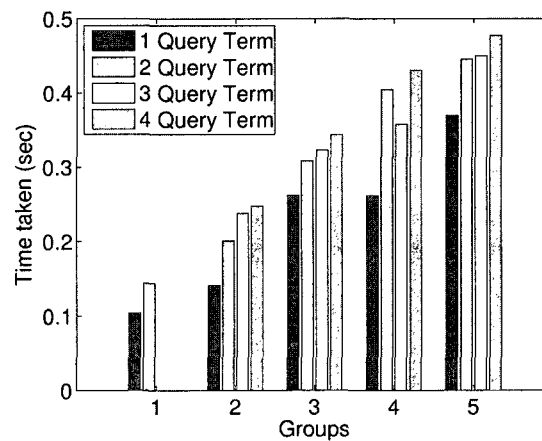
#### **4.6.7 Alternative Design**

An alternative system design is to not use an inverted index at all. The incoming metadata is buffered and flushed to flash when there are enough entries to make up a full metadata page. Each



Groups	1 term	2 terms	3 terms	4 terms
1(0.1s)	16	156	x	x
2(0.2s)	8	16	68	156
3(0.3s)	6	11	16	46
4(0.4s)	4	8	12	16
5(0.5s)	4	7	10	13

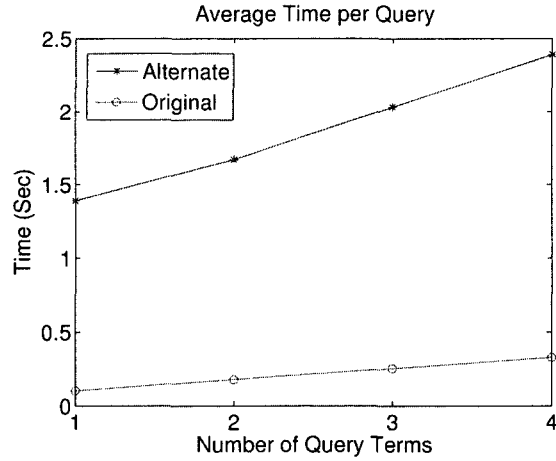
**Table 4.3:** Recommended size of main index ( $H$ ) for different query response requirements .



**Figure 4.11:** Actual query response time.

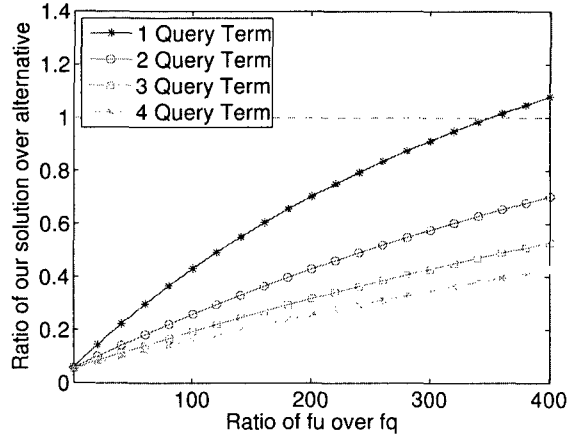
metadata page will contain a pointer to the previous metadata page in flash. A single entry kept in memory remembers the latest metadata page's location in flash. When querying, Microsearch accesses every metadata page in flash before replying since every metadata page could contain a payload matching the query terms. The intuition is that such a scheme will have a better indexing performance at the expense of worse query performance.

To evaluate, we used a 3KB memory limit. The alternative design will allocate all as much space as possible to the buffer cache, and have just one main index entry. Microsearch uses a



**Figure 4.12:** Comparing alternative scheme with our scheme

balanced approach, using an inverted index size of  $H = 32$  (96 bytes), and a buffer cache of  $B = 372$  (2976 bytes). The alternative system takes an average of 6.5 ms to insert the metadata in one file compared to the 17.5 ms for our scheme. Fig. 4.12 shows the difference in query response time for different number of query terms. Next, we compare the energy consumption between our scheme and the alternative scheme. Since both schemes have to do the same amount of writing for the payload data given the same document set, our comparison only measures the energy consumption of metadata input and query. Let  $P_w$  and  $P_r$  be the energy consumption for writing and reading one page data in flash memory respectively. Given the input insertion frequency  $f_u$  and user query frequency  $f_q$ , the energy consumption is determined by the amount of metadata writing during the input insertion period and the amount of metadata reading during the query period. For the simplicity, we ignore the energy consumption of CPU processing because that part is much smaller compared with the flash memory read and write operations. On a per unit time basis, the energy consumption of our scheme can be expressed as  $E_1 = f_u \cdot (W_i \cdot P_w + R_i \cdot P_r) + f_q \cdot R_q \cdot P_r$ , where  $W_i$  and  $R_i$  are the numbers of write and read operations for insertion, and  $R_q$  is the number of reads required for the query. From Section 5, we have  $W_i = \lceil \frac{D \cdot m}{E'} \rceil$ ,  $R_i = \frac{D \cdot m}{x}$ , and  $R_q = \frac{2 \cdot D \cdot m \cdot t}{E' \cdot H}$ .



**Figure 4.13:** Comparing power consumption of our scheme verses alternative scheme

For the alternative scheme, we consider a more efficient insertion process without reading the last flushed metadata pages. Instead, new metadata pages will be directly written to the flash. Therefore, the energy consumed by the alternative scheme can be expressed as  $E_2 = f_u \cdot W'_i \cdot P_w + f_q \cdot R'_q \cdot P_r$ , where  $W'_i = \lceil \frac{D \cdot m}{E} \rceil$  and  $R'_q = \frac{2 \cdot D \cdot m \cdot t}{E}$ . With the system parameters fixed at  $D = 622$ ,  $m = 3.3$  and  $H = 32$ , we estimate the energy consumption for both schemes based on TelosB flash memory read and write energy performance presented in [76] (i.e.,  $P_w = 0.127 \times 256 = 32.5 \mu J$ ,  $P_r = 0.056 \times 256 = 14.3 \mu J$ ).

To compare our scheme with the alternative, we find the ratio of  $\frac{E_1}{E_2}$ . Values less than 1 favor our solution while values larger than 1 favor the alternative. To simplify the results, we divide both  $E_1$  and  $E_2$  by  $f_q$ , which does not affect the ratio. As a result,  $\frac{E_1}{E_2}$  becomes a function of  $\frac{f_u}{f_q}$ . We plot the energy ratio graph with 1, 2, 3 and 4 query terms respectively. The estimation results are found in Fig. 4.13. The figure shows that for an average of 1 query term, the alternative performs better when there are about 350 document insertions to a single query. For other cases, our scheme is always superior to the alternative solution. This suggests that the alternative scheme should be used only when the mote is used to store data and rarely if ever queried.

## 4.7 Conclusion

In this chapter, we present a search system for small devices. Our architecture can index an arbitrary number of textual metadata efficiently. A space saving algorithm is used in conjunction with IR scoring to return the top- $k$  answers to the user. Our experimental results show that Microsearch is able to resolve a user query of up to four terms in less than two seconds, and provide a high level of accuracy.

---

**Algorithm 8** Reply Top- $k$  Query:

---

```

1: Input:  $k, \{st_1, st_2, \dots, st_t\}$ 
2: Output:  $k$ -length array result
3:  $head[i] = InvIndex[hash(st_i)]$ 
4: Scan buffer and each relevant metadata page chain to accumulate the document frequency
   ( $df[i]$ )
5: Load relevant index entries in buffer to the buffer page  $page[i]$ 
6: If  $page[i]$  is empty, load  $Flash(head[i])$  and move  $head[i]$  to the next page
7: while there exists a non-empty  $page[i]$  do
8:    $cutoff = \max\{\min(page[i]), \forall i \in [1, t]\}$ 
9:   for non-empty  $page[i]$  and  $\max(page[i]) \geq cutoff$  do
10:    for every entry  $e \in page[i]$  and  $e \geq cutoff$  do
11:       $score = calScore(e)$ 
12:      if  $score >$  minimum score in result then
13:        replace the entry with the minimum score in result by  $\{e, score\}$ 
14:      end if
15:      for  $j = 1$  to  $t$  do
16:        remove  $e$  from  $page[j]$ 
17:      end for
18:    end for
19:  end for
20:  for  $i = 1$  to  $t$  do
21:    if  $page[i]$  is empty then
22:      load  $Flash(head[i])$  to  $page[i]$ 
23:      move  $head[i]$  to the next metadata page
24:    end if
25:  end for

```

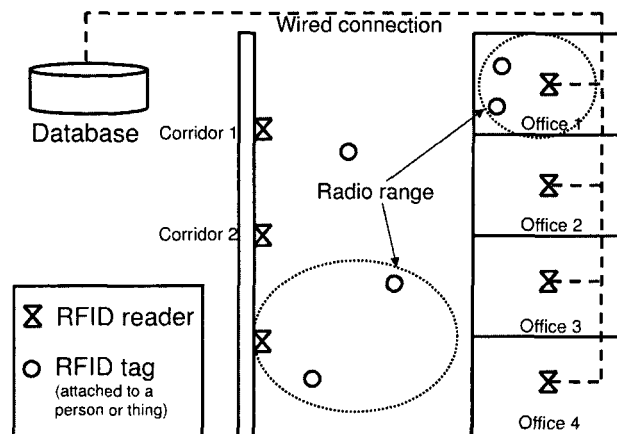
## Chapter 5

# Privacy Protection for RFID-based Tracking Systems

In this chapter, we depart from local access to consider remote access for an RFID tag. The motivation behind examining remote access is to understand the modifications needed to ensure the security of the extracted data. In this chapter, we use an RFID-based tracking system to illustrate how we can generate a tag reply that remains secure even when the aggregated data is partially compromised.

The low cost and rugged design of RFID tags make them suitable to be attached to everyday objects such as key chains and coffee mugs. With every object embedded with its own RFID tag, ubiquitous computing concepts like object tracking and localization or location based services can be realized [48, 100], allowing many new applications to be developed upon this infrastructure. At the same time, widespread RFID deployment creates privacy risks for everyone, not just a few willing early adopters, thus the need for privacy protection becomes critical.

Fig. 5.1 shows an environment where RFID tags are attached to people and objects. Each



**Figure 5.1:** Typical RFID tracking system

RFID tag has an associated unique ID which represents that tag in the system. A network of RFID readers are deployed in the building, with each reader associated with a specific location, such as “Office 1” or “Corridor 2”. The readers are connected to a database server via a wired network. A reader will periodically broadcast a query, and all the RFID tags within the vicinity will respond to that reader. The reader will then forward these tag responses to the database. A user wanting to know for instance who was at a meeting in Office 1 at 1:00 pm can query the database to determine the IDs of the tags, and thus the identity of the meeting participants.

The potential for abuse of such a powerful tracking system is apparent. The straightforward approach is to limit the access to the database data using passwords. Each user is associated with a list of RFID tags they are authorized to access, and must supply the correct password when querying the database. For example, a user with ID=101 running a query “SeLect location From database Where ID=101 and Time=time” must first supply the appropriate permissions for his tag before the location information is released. More advance techniques such as role-based access control [16, 89], and Hippocratic databases [3], can also be used to protect the database.

This type of solutions have a common requirement. **A trusted database is needed to main-**

**tain and enforce the privacy policy.** Without a database that is invulnerable to compromise by criminal hackers or disgruntled employees, and always managed by competent system administrators, we cannot protect user privacy. We believe that maintaining such a trusted server is difficult in practice. Even three letter government agencies with strong emphasis on security have been vulnerable to database breaches or mistakes [1, 31, 38].

In this chapter, we consider the problem of how to build a tracking system that does not rely on a trusted database to protect user privacy. Our solution divides up the database operations into separate servers such that we can tolerate the compromise of any one of the databases. Our protocols are designed to take into account the computationally weak capabilities of an RFID tag, and allow a user to efficiently query the database for answers.

We make the following contributions in this chapter. We propose a technique to divide a database query, originally intended to a single database, into multiple databases such that if only one of the database is compromised, the user's privacy is protected. We also consider how to detect, but not prevent, adversary disruption attacks such as the data deletion and manipulation.

## **5.1 Related Work**

Privacy protection is an important component in ubiquitous computing environments [4, 66]. The technique of using mix zones was proposed by [14, 15] where by users could specify certain areas where nobody could trace their movements. Other researchers [49, 52, 81] proposed techniques to help users define privacy policies. This approach is more flexible than mix zones at the cost of additional complexity in specifying the policies. The idea of allowing users to specify “virtual walls” was suggested by [60] to simplify the creation of a privacy policy. Physical access control [64, 90] addresses the problem of specifying a privacy policy. Users can only obtain the



location information of people that are were present together at the same time. The intuition is that users that were at the same location at the same time already know each other's presence, and thus there is no privacy issues when releasing that information later. Our work differs from these proposed techniques in that we do not rely on trusted servers to protect user privacy. Our idea of separating location, time, and identity is similar to that proposed by [94], but our solutions are designed to work with RFID tags.

RFID security is an active area of research with many different protocols being proposed [79, 83, 118]. While our work also proposes a simple security protocol, our focus is less on the security and privacy between RFID reader and tag, but oriented more towards data already collected and archived.

Closely related to our work is research on searching encrypted data. In this problem, a user encrypts his data and stores it at an untrusted server. The user wants to be able to search of part of his data in an efficient manner. Since the server is untrustworthy, the user cannot send over his secret key. The user also cannot request the server to transmit all the encrypted data back since it is inefficient. An search system using symmetric key to encrypt data was proposed by [99], while [19] suggested a public key based scheme. Practical encrypted database query retrieval systems were proposed by [116, 120]. However, unlike our solution, prior research in this area do not consider the privacy implications of ubiquitous environments such as malicious tracking of users. This was shown in the second strawman approach by using [120] as an example. Furthermore, these prior techniques assume that more advance hardware such as laptops are used, rather than computational weak RFID tags.

## 5.2 Problem Formulation

We consider a network of RFID readers,  $R_1, \dots, R_n$ , are deployed throughout a facility. Each reader is programmed to periodically broadcast a query to read all the RFID tags within its vicinity. The captured data is then forwarded to a backend database for storage. In the unencrypted scenario, the database stores this information as a  $ID : time : location$  tuple, where ID is a number that identifies that RFID tag, time is the time that tag was read, and location is the physical location corresponding to that particular reader. We assume that the database knows the locations of all the RFID readers, and can associate the data from a reader to a location.

The database can represent the data from all the RFID readers as a table with 3 attributes, ID, Time, and Location. Table 5.1 illustrates this database. A user who own tag 101, can query this

**Table 5.1:** Unencrypted table in database

	ID	Time	Location
1.	101	10:00am	Office 1
2.	102	10:00am	Office 3
3.	101	10:15am	Office 2
...	...	...	...

database later by issuing a query,

```
Select * from DB where ID=101 and Time=10:00 am
```

and determine he was at Office 1 at 10 am.

Such a setup provides no privacy protections, since anyone can query the database to determine anybody's movements. Under the trusted server assumption, we can protect privacy by associating

a password with each ID. The user running the above query for instance, will have to supply the correct password associated with ID=101, before the database will release the information. While more complex schemes can be designed to provide better access control, a fundamental requirement is that the database *cannot* be compromised. An adversary with access to Table 5.1 learns everything. Here, we consider the problem of how to protect privacy in the event of such a server is compromised by an adversary.

### 5.2.1 Adversary model

We assume that the adversary seeks only to track the movements of a user. The adversary succeeds if he is able to extract the identity of the user from the database, or if he is able to link two entries in the database to the same user.

We assume that the adversary can have free access to the database data such as in Table 5.1, as well as observe the database interactions between a user and the database. The adversary is however, unable to determine the identity of someone querying the database. For instance, the adversary cannot deduce a user identity through the MAC address of the user's device when the user is querying the database. Techniques such as an anonymizing network [35] can be deployed to achieve this. Also, the adversary cannot reprogram the database to execute functions it otherwise will not perform.

Here, we assume that the RFID tags are able to perform simple operations such as generating random numbers, perform a hash function, and XOR two bitstrings together. These are common assumptions made in RFID security literature [7, 23, 85]. While our solution in the paper is presented using hash functions, symmetric key encryption can be substituted instead with minor modifications.

Finally, we assume that having a rational adversary precludes attacks such as deleting or shuffling entries in the database since such actions do not help the adversary identify a user. This is reasonable since such disruption attacks increases the risk of detection. We will discuss more about defending against such attacks in the Section IV.

### 5.3 Strawman solutions

We motivate our solutions by considering the limitations of seemingly possible solutions. For all these strawman solutions, we assume a more powerful RFID tag that can do symmetric key encryption is used. We always assume the user is the owner of RFID tag 101.

#### 5.3.1 Strawman 1

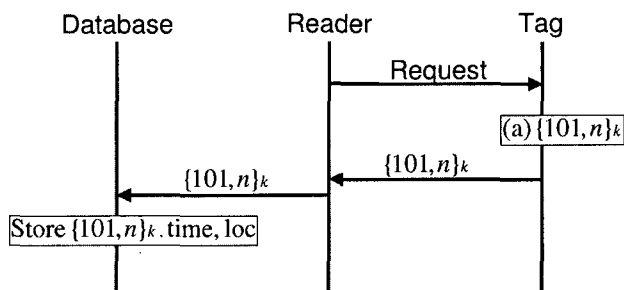


Figure 5.2: Strawman protocol 1. The tag ID is 101.

In this simple protocol, we let the RFID tag return its encrypted ID in the form of  $\{ID, n\}_k$  where  $n$  is a random number, and  $k$  is the tag's secret key. For completeness, we show this interaction in Fig.5.2. Now, in the database, we have

where  $k_1$  and  $k_2$  are the secret keys of tags 101 and 102 respectively. We see that this approach protects privacy since the adversary observing this table cannot learn the actual IDs, 101 and 102,

**Table 5.2:** Database table from Strawman 1

	ID	Time	Location
1.	$\{101, n_1\}_{k_1}$	10:00am	Office 1
2.	$\{102, n_2\}_{k_2}$	10:00am	Office 3
3.	$\{101, n_3\}_{k_1}$	10:15am	Office 2
...	...	...	...

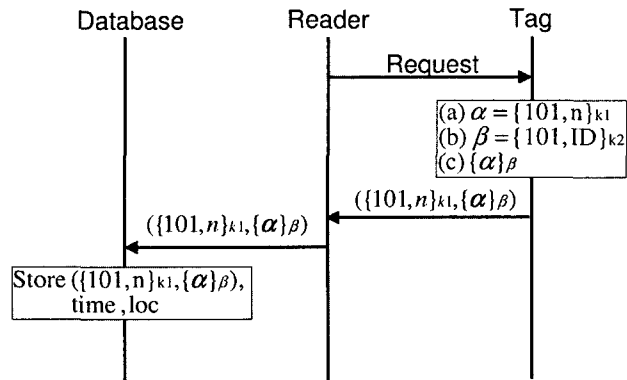
of the tags. There is also no linkability, since  $\{101, n_1\}_{k_1}$  and  $\{101, n_3\}_{k_1}$  use different random numbers  $n_1$  and  $n_3$  while encrypting the same ID, thus resulting in different ciphertexts.

The problem with this solution arises when the user wants to query the database. He can no longer simply do a “Select \* from DB where ID=101”, since all the ID attributes now have a random number component. The user cannot recall the  $n_1$  and  $n_3$  values used since the limited storage capacity of the RFID tag means these random numbers have to be generated on-the-fly and never archived. The only option is for the user to retrieve the *entire* table, and attempt to decrypt each entry’s ID field until he finds the all the entries with IDs equal to his own. This is clearly inefficient given the large size of the table.

### 5.3.2 Strawman 2

We can modify a technique from [120] to improve the query performance. Now, each RFID tag maintains two keys that it keeps secret, i.e. tag 101 will have  $k1^{101}, k2^{102}$ . We remove the superscript and use  $k1$  and  $k2$  to denote keys to simplify the presentation. The protocol is shown in Fig. 5.3, and after the data is collected, the table resembles Table 5.3.

This approach also provides the same protections against an adversary as the earlier strawman protocol, since knowing  $\{101, n_1\}_{k_1}, \{\alpha_1\}_{\beta_1}$  does not lead to knowing 101. Furthermore, the ad-



**Figure 5.3:** Strawman protocol 2. The tag ID is 101.

**Table 5.3:** Database table by Strawman 2

	ID	Time	Location
1.	$\{101, n_1\}_{k1}, \{\alpha_1\}_{\beta_1}$	10:00am	Office 1
2.	$\{102, n_2\}_{k1}, \{\alpha_2\}_{\beta_2}$	10:00am	Office 3
3.	$\{101, n_3\}_{k1}, \{\alpha_3\}_{\beta_1}$	10:15am	Office 2
...	...	...	...

versary cannot link  $\{101, n_1\}_{k1}, \{\alpha_1\}_{\beta_1}$  and  $\{101, n_3\}_{k1}, \{\alpha_3\}_{\beta_1}$  together since a different random  $n$  is used.

When a user queries the database, he will issue a “Select \* from DB where ID= $\hat{\beta}$  and Time=10:00am”, where

$$\hat{\beta} = \{101, ID\}_{k2}$$

The database will encrypt the first portion of each ID field with  $\hat{\beta}$ , and check whether it matches the second portion. If it does, the database will return that entry to the user. For example, encrypting  $\{101, n_1\}_{k1}$  with  $\hat{\beta}$  will match  $\{\alpha_1\}_{\beta_1}$ , and hence this entry belongs to the user. Since  $\{101, n_1\}_{k1}$  is protected via  $k1$  which is only known by the user, the database does not have to verify the user.

It is clear that strawman 2 is more efficient than strawman 1.

However, once an adversary observes  $\hat{\beta}$ , the adversary can determine future time and locations that  $\hat{\beta}$  have visited. For instance, using  $\hat{\beta}$ , the adversary know that the same tag visited Office 2 at time 10:15 am. The reason is that while at time 10:15 am we have  $\{101, n_3\}_{k1}$  which uses a different  $n_3$ , the same  $\beta$  is used, since  $\beta = \{101, ATTR\}_{k2}$ . The  $ATTR$  is a fixed value in the database and cannot be changed by the user. Thus, strawman 2 only prevents linkability so long as the adversary *never* observes a user querying the database.

This vulnerability does not occur in [120] because in their encrypted database, all fields are encrypted by the user and stored into the table. In other words, “10:15 am” and “Office 2” are all encrypted separately. The “ $\beta$ ” used to encrypted “10:00 am” and “10:15 am” will be  $\{10:00 \text{ am}, Time\}_{k2}$  and  $\{10:15 \text{ am}, Time\}_{k2}$ , so knowing the  $\beta$  value of 10:00 am does not reveal anything about 10:15 am. We cannot do the same in an RFID based tracking system because the RFID tag **cannot** determine the time and location independently.

### 5.3.3 Discussion

From the strawman protocols, we can make the following observations. First, a direct encryption of data by the RFID tag (strawman 1) protects user privacy even if the database is compromised. However, this approach has slow query performance since the database cannot return only the user’s data to him. Second, the solution cannot merely focus on building an encrypted database, but must also consider the user query process. The protocol has to ensure that the user’s query does not reveal useful information that an adversary can use to violate user’s privacy (strawman 2). Finally, unlike more powerful laptop devices [93], the RFID tag cannot maintain an internal clock to determine time by itself, not capture IP address or GPS coordinates to determine its

location independently. While [107] gives a solution for the RFID tag to determine time, it is unclear that the same techniques can be applied to location information since the tag movements is unpredictable.

The intuition behind our approach is to utilize separate database servers to perform different roles in the RFID tracking system, as well as store different types of data in each server. There are two roles in the RFID tracking system, where the tag was read (location), and at what time the tag was read (time). Our approach uses two database servers, one to control and store time data and the other to control and store location data. **Here, we assume that the adversary can only compromise one of the following, (a) the database storing time information, (b) the database storing location information, (c) a small number of RFID readers.**

We justify this assumption because the two servers can be housed at different physical locations, running different operating systems, and managed by a separate group of system administrators. This raises the bar for an adversary to compromise both servers. Also, given the large number of RFID readers deployed, it is reasonable to assume that the adversary cannot control a majority of them.

The following notations are used to describe our protocols. The server controlling the timestamps is the timestamp server  $TS$ , and the server controlling the location is the location server  $LS$ . We use terms server and database interchangeably. While both  $TS$  and  $LS$  can communicate with the RFID readers, only the  $LS$  knows the location of these RFID readers. We denote an RFID reader as  $R$ , and an RFID tag as  $T$ . Each  $T$  maintain its own secret  $s$ ,  $ID$  and a simple incremental counter  $ct$ . The values of  $s$ ,  $ID$ , and  $ct$  are kept secret and only known to the tag owner. The tag also maintains a simple incremental counter  $ct$ . We use subscripts  $i$  when denoting more than one reader or tag.



We denote the RFID tag identifier as  $\omega$ , and  $\omega$  will be stored under the “ID” attribute in the *LS*. For instance, when there is no security at all, the  $\omega$  value is just *ID*, in strawman protocol 1, the  $\omega$  value will be  $\{ID, n\}_k$ , and in strawman protocol 2,  $\omega = (\{101, n\}_{k1}, \{\alpha\}_\beta)$ . A summary of the notation used is given in Table 5.4.

**Table 5.4:** Summary of Notations

<i>R</i>	RFID reader
<i>T</i>	Tag
<i>TS</i>	Timestamp Server
<i>LS</i>	Location Server
<i>s</i>	Tag’s secret, known only to tag owner
<i>ID</i>	Tag’s ID, known only to tag owner
<i>ct</i>	Tag’s counter value, known only to tag owner
$\omega$	Tag identifier
<i>t</i>	timestamp, stored in TS
<i>loc</i>	location, stored in LS
<i>n</i>	Nonce generated by tag
$N_{ls}$	Nonce generated by LS

## 5.4 RFID Protocol

The intuition behind our protocol is for the RFID tag to generate two pieces of data each time it responds to an RFID reader. The reader will store one piece of data associated with the time the tag was read in the *TS*, and the other piece of data associated with the location of the tag in the *LS*.

An adversary compromising either *LS* or *TS* will be unable to violate the privacy of the RFID tag. When a legitimate user wishes to query the databases, he will query both *TS* and *LS* separately and combine the result to satisfy his query. The overview of an RFID reader collecting data and user querying the servers is shown in Fig. 5.4.

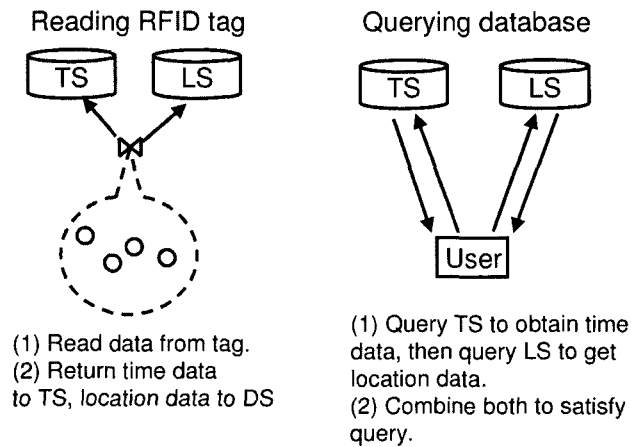


Figure 5.4: Obtaining data from tags, and querying databases for data.

### 5.4.1 Collecting data from tag

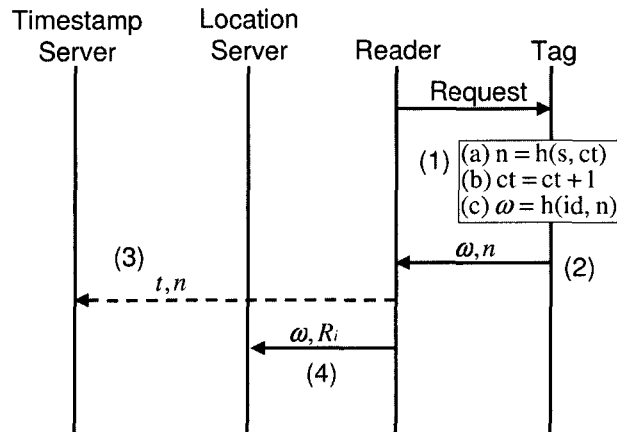


Figure 5.5: Reader-tag interaction. The dotted line in step (3) denotes that the reader transmits directly to the timestamp server, bypassing the location server.

Fig. 5.5 shows the protocol for collecting data from an RFID tag. When a reader queries the RFID tag in Step (1), the tag will first generate a random number  $n$  by hashing its secret  $s$  with the current counter value,  $ct$ . The tag will then increment the  $ct$  by one (Step 1b), and create the identifier  $\omega$  as  $h(ID, n)$ . Finally, in Step (2), the tag returns  $\omega, n$  to the reader.

When the reader receives this tuple, the reader will append the current timestamp  $t$  together with  $n$ , and send that to the  $TS$  in Step (3). This messages bypasses the  $LS$  completely, meaning the  $LS$  never learns  $n$ . Then, the reader will append the reader's ID,  $R_i$ , and  $\omega$ , and send everything to the  $LS$  in step (4). Using  $R_i$ , the  $LS$  can determine the reader's.

**Table 5.5:** Table maintained by  $TS$

	Time	Random value
1.	10:00 am	$n_i$
2.	10:00 am	$n_j$
3.	10:15 am	$n_k$
...	...	...

At the end of the protocol, the  $TS$  maintains a table shown in Table 5.5. The  $TS$  table has 2 attributes, time, and a random value. The random values associated with time 10:00 am for instance, are all the  $n$  values transmitted by all the RFID readers. Each  $n$  value represent a different RFID tag. Similarly, the  $LS$  maintains a table shown in Table 5.6. The  $LS$  table has 2 attributes, the tag identifier  $\omega$  and the location that tag was read. Each entry in the  $LS$  table represent a different RFID tag response. Note that the  $LS$  does not update the table in real time, and thus the ordering of entries do **not** indicate the time an RFID tag was read.

**Table 5.6:** Table maintained by *LS*

	ID	Location
1.	$\omega_1$	Office 1
2.	$\omega_2$	Office 3
3.	$\omega_3$	Office 2
...	...	...

### 5.4.2 Querying the database

Let the user wanting to learn his location at 10:00 am. He will need to query *LS* using the  $\omega$  value  $h(101, n)$ , where  $n$  is the value he choose at 10:00 am. Notice that the *LS* table is similar to strawman protocol 1's table. In both instances, the ID 101 is protected by a random number  $n$ . To query *LS*, the user must first determine the value of  $n$  he used at 10:00 am. In our protocol, we let  $n = h(s, ct)$  and increment  $ct$  immediately afterwards. While the tag cannot recall the random  $n$  value used at 10:00 am, the owner of the tag can determine the *current* value of  $ct$  in the tag. (We assume that there is a command to recover this.) Knowing the secret  $s$  as well as the initial and current value  $ct$ , the user can re-generate all the random  $n$  values the RFID tag has used so far. For instance, the user can generate a list The user then queries the *TS* database using

$$\text{Select Time from } TS \text{ where Random Value} = n_i, \quad (5.1)$$

where  $n_i = h(s, ct_i)$ .

After receiving the time corresponding to  $n_i$  from Table 5.5, the user determine whether the returned time is larger than or smaller than his target time of 10:00 am. If the returned time is larger, the user will pick another an smaller  $ct$  value, compute  $n$ , and query the *TS* again. If the

**Table 5.7:** Table maintained by user

ct	n	time
...	...	...
$ct_{i-1}$	$h(s, ct_{i-1})$	?
$ct_i$	$h(s, ct_i)$	?
$ct_{i+1}$	$h(s, ct_{i+1})$	?
...	...	...

returned time is smaller, the user will pick a target  $ct$  value. Using a simple binary search, the user only needs to execute  $O(\log n)$  queries to  $TS$ .

With the user now knowing his  $n$  corresponding to 10:00 am, he can now query the  $LS$  server using the following query.

$$\text{Select } * \text{ from } LS \text{ where ID}=\omega, \quad (5.2)$$

where  $\omega = h(101, n)$ .

### 5.4.3 Security analysis

We consider the security of our system after the adversary compromises the  $TS$ , the  $LS$  and the reader  $R_i$ . Since we allow anyone to query the databases, the adversary controlling the  $TS$  for instance, can also query the  $LS$  like a regular user.

**Compromise  $TS$  server:** The adversary succeeds in attacking  $TS$  if he can determine which  $\omega$  in  $LS$  belongs to a user, or if the adversary can determine that two  $\omega$  values belong to the same person. The reason for focusing on  $\omega$ s is because through  $\omega$ , the adversary determine the whereabouts of a user.

The adversary controlling *TS* is able to access all the records such as those in Table 5.5, as well as observe multiple queries (Query 5.1) made by a user and the corresponding response. The adversary can also query the *LS* using information from his observations.

We begin by examining what the adversary can learn from controlling *TS*. For a single RFID tag  $T_i$  with secret  $s_i$  being queried twice, the  $n$  results from  $h(s, ct)$  and  $h(s, (ct + 1))$  will be different since the  $ct$  value is automatically incremented each time the tag responds, as shown in Fig. 5.5 Step 1(b). We see that the adversary simply knowing the entire Table 5.5 cannot determine whether two  $n$  values belong to the same user or not. The adversary can only determine which  $n$  values belong to the same user after observing a user execute Query 5.1 multiple times. The reason is that the answer to Query 5.1 is the  $n$  value associated with a particular time. Since only a user knows his own  $s$  values, successive Query 5.1 link the  $n$  values to the same user.

Let us assume the adversary after some observation can determine that the times  $t_i$  and  $t_j$ , and the corresponding  $n_i$  and  $n_j$  belong to the same user. Now the adversary tries to query *LS* to try and determine where this user has been. The table maintained by *LS* only contains a set of  $\omega$ s and their corresponding locations. There is no indication *what* time each  $\omega$  was obtained. The adversary cannot determine which  $\omega$  belongs to the user he is tracking because  $\omega = h(ID, n)$ , and that the adversary cannot link  $\omega = h(ID, n_i)$  and  $\omega = h(ID, n_j)$  together without knowing the secret  $ID$ .

The property of separating time and location information into the *TS* and *LS* respectively defends against leaking information in the more extreme instances where there are few users in the entire tracking system. Consider the tracking system of an office building at night, and we have *TS* table,

	Time	Random value
1.	2:00 am	$n_i$
2.	2:15 am	$n_j$

Assuming there is no one else in the building, the adversary can infer that  $n_i$  and  $n_j$  belong to the same person. Now, the adversary can attempt to query  $LS$  to determine where that person has been. The adversary cannot determine any  $\omega$ s from  $n_i$  and  $n_j$ , and can only issue a query “Select \* from  $LS$  where ID=\*” to retrieve *everything* from  $LS$  to try and determine where this tag has been. Since  $LS$  does not store time, the adversary cannot filter the  $LS$  data to narrow down possible locations the tag has been.

**Compromise  $LS$  server:** Next we consider the adversary controlling the  $LS$  server. The adversary will now be able to access all the records such as those in Table 5.6, as well as observe multiple queries (Query 5.2) made by a user and the corresponding response. The adversary can also query the  $TS$  using information from his observations. The goal of the adversary remains the same.

From controlling  $LS$ , the adversary knows the time and location associated with each  $\omega$ . Given that  $\omega = h(ID, n)$ , RFID tags with different IDs will have different  $\omega$  values, and the same tag will also have different  $\omega$ s at different times since the  $n$  values will change due to  $n = h(s, ct)$ , and the tag’s  $ct$  values increments each time it replies. Unlike controlling the  $TS$ , the adversary observing multiple Query 5.2 cannot assume they all belong to the same user and link the  $\omega$ s together. This is because a user does not have to issue Query 5.2 more than once to obtain an answer.

Let us assume that the adversary knows that  $\omega_i$  is associated with time  $t_i$ . The adversary can query  $TS$  doing “Select \* from  $TS$  where Time= $t_i$ ”. However, since there are many tags that respond at each time, all the adversary obtains is a set of  $n$  values. The adversary cannot determine

which  $n$  value corresponds to his  $\omega_i$ .

**Compromise RFID reader:** An adversary controlling an RFID reader will be able to read  $\omega, n$  from a tag (Step (2) in Fig. 5.5), and is assumed to know the location of the reader it has compromised. We allow the adversary to physically observe a user transmitting a particular  $\omega_i, n_i$  once, in other words, being able to associate a user's identity to a  $\omega_i, n_i$  tuple. The adversary succeeds if he is able to use this information to determine additional information regarding that user.

One attack has the adversary trying to use  $\omega_i, n_i$  in querying  $TS$  and  $LS$ . The adversary learns from querying  $TS$ , since he already knows the time and  $n_i$  values. Since the tag will use a different  $n$  each time, the adversary cannot determine whether other  $n$  values belong to the same user. Similarly, since the  $n$  value is constantly changing, the adversary observing  $LS$  cannot use  $\omega_i$  to determine which  $\omega$ s belong to the same user. Thus, no additional information can be obtained from  $TS$  or  $LS$  from  $\omega_i, n_i$ .

Another attack has the adversary after manually determining the identity associated with  $\omega_i, n_i$ , trying to determine if a future  $\omega_j, n_j$  belongs to the same user. This is useful if the adversary controls the reader deployed outside an sensitive location like a clinic. Since the adversary cannot always be physically present to determine a user's identity, this attack allows the user to determine if the same user has visited that location again in the future. However, since knowing the  $n_i$  and  $n_j$  cannot be linked together because the  $ct$  value is incremented and hashed with a secret  $s$  known only the tag, the resulting  $\omega_i$  and  $\omega_j$  cannot be linked together.

Finally, we consider the scenario where the adversary controls multiple RFID readers. Controlling multiple readers does not give the adversary any additional advantage, since a tag does not



have to authenticate the RFID reader before transmission. The tag will always generate a different  $\omega, n$  tuple to any reader that queries it.

#### 5.4.4 Protocol discussion

Our protocol uses the counter  $ct$  that automatically increments each time the tag is queried. This feature allows an adversary, using his own reader, to query the tag simply to increment the  $ct$  value. However, this behavior only degrades the user's performance, and does not help the adversary learn anything about the user. A rational adversary will not launch this type of attack.

Our choice of  $\omega$  in the protocol is  $h(ID, n)$ . Given an adversary, a possible alternative is to set  $\omega$  as  $h(ID, t)$ . This will have the same properties as  $h(ID, n)$  since the time value  $t$  will only occur once and never repeat. Using  $h(ID, t)$  will also give better query performance, since the user can directly determine the appropriate  $\omega$  and query the  $LS$ , instead of doing a binary search on  $TS$  to determine  $n$ . The reason we do not use  $h(ID, t)$  is that different readers may have a slightly different clock skew. Thus, honest readers may issue the same  $t$  value to the RFID tag, resulting in similar  $\omega$  values at different locations. This allows that tag to be linked to two locations, thus violating privacy. Our choice of  $h(ID, n)$  does not have this problem since the  $ct$  will automatically increment after each query, resulting in different  $\omega$  values each time.

### 5.5 Additional Discussion

Here we consider disruption attacks that do not impact user privacy but can disrupt regular user operations.

### 5.5.1 Detect deleted data

The adversary controlling either  $TS$  or  $LS$  can decide to delete selected entries from the respective tables. The adversary can do this simply to disrupt the database operations, or to conceal other malicious activities. Consider for instance an adversary planning to steal something from an office. The adversary can attack the  $TS$  or  $LS$  to avoid storing any tag data collected around that time or location to cover his tracks. Possible witnesses that check the system to verify their locations will determine that they were not actually present at that time.

Recall that when a user wishes to query “Select \* from  $LS$  where  $ID=\omega$ ”, he will first build a Table 5.7 to determine his target time. To determine whether any data has been deleted, the user can expand Table 5.7 to include enumerate all previous  $ct$  values until up to the current  $ct$  value in his RFID tag. He then queries the  $TS$  until to determine the  $n$  value corresponding to each  $ct$ . If the adversary has compromised  $TS$  and deleted his entry at a particular time, the  $n$  value corresponding to that time will be missing. In other words, if the user has some counter value  $ct_i$  where there is no  $n$  value in Table 5.5 matches  $h(s + ct_i)$ , the user will suspect that his data has been deleted.

If the user can obtain all the  $n$  values corresponding to his  $ct$  values, he then queries the  $LS$  for each  $\omega$  associated with each  $n$ . If the adversary has deleted his entry from  $LS$ , the user will not receive any location associated with one of his  $\omega$ s. Note that the adversary can only select entries to delete based on either time, if controlling  $TS$ , or location if controlling  $LS$ . The adversary cannot single out a particular tag’s information to be deleted since he cannot distinguish between two tags.

The adversary can compromised the RFID reader instead of  $TS$  or  $LS$  so that the reader does not broadcast any requests. When this happens, no tag data exists in either  $TS$  or  $LS$ , and the RFID

tag will not be triggered increment its  $ct$  value. While a user can infer from “gaps” in the time and location information from  $TS$  and  $LS$  that a particular RFID reader might be compromised, the user cannot be certain since the gaps may also be caused by environmental conditions or faulty readers. Nonetheless, the occurrence of such gaps will trigger an investigation and detect any compromised readers.

### 5.5.2 Detect tampered data

Instead of deleting the data, the adversary can choose to tamper with the time or location information and launch an attack as follows. Consider an RFID tag at 10:00 am was read outside “Office 1”. There will be an entry in the  $TS$

	Time	Random value
1.	10:00 am	$n$

where  $n = h(s + ct)$ . The corresponding entry in  $LS$  will be

	ID	Location
1.	$\omega$	Office 1

Now let the adversary compromises  $TS$ , and changes the time variable from 10:00 am to 11:00 am. The user querying  $TS$  with “Select \* from  $TS$  where Random Value =  $n$ ”, will receive the answer 11:00 am. The same user now querying  $LS$  with  $\omega = h(ID, n)$  will believe that he was outside Office 1 at 11:00 am instead of 10:00 am. Since no data was deleted, the user using the technique for detecting missing data above will not find any problems. The adversary that compromises  $LS$  can execute the same attack by changing “Office 1” to “Office 2”.

The reason this attack is successful is because there is nothing linking the value  $n$  to 10:00 am, nor the value  $\omega$  to “Office 1”. We can modify our protocols to let the RFID reader transmit both

the  $t$  and  $R_{ID}$  information when querying a tag. The tag will then compute a new variable  $\varepsilon$  where

$$\varepsilon = h(ID, n, t, R_{id}).$$

and return this value to be stored in  $LS$ . Thus, the table in  $LS$  will become

	ID	Location
1.	$\omega, \varepsilon$	Office 1

After the user queries for his location from  $LS$ , he will compute  $\hat{\varepsilon}$  by hashing his ID with the  $n$  value and time values he received from  $TS$ , and the location provided in  $LS$ . If  $\hat{\varepsilon}$  matches  $\varepsilon$ , the user will accept the answer.

We can use a separate mechanism to detect whether an adversary has compromised an RFID reader to an incorrect  $t$  or  $R_{ID}$  values. When  $LS$  receives the data from an RFID reader, it can check whether the contained  $R_{ID}$  matches the RFID reader IP address that transmitted the data. A warning will be flagged if there is a discrepancy. The same check can be performed by  $TS$  to verify if the reader used an incorrect  $t$  value.

The adversary controlling  $TS$  can use this  $\varepsilon$  to determine the location of user if he is able to associate an  $n$  value with  $\varepsilon$  since he will then be able to search  $LS$  for a matching  $\varepsilon$  value. The adversary cannot obtain  $\varepsilon$  directly since this value is never forwarded to  $TS$ . The adversary cannot deduce this value from  $n$  and  $t$  because  $\varepsilon$  requires knowing  $R_{ID}$  and ID, both which the adversary does not know.

For the adversary controlling  $LS$ , the addition of  $\varepsilon$  can be used to track a user if the adversary can observe the  $LS$  table and determine two identical  $\varepsilon$  values. This will imply that the same user visited both locations. However, each  $\varepsilon$  contains a time  $t$  from the RFID reader which will never

repeat itself, and the adversary controlling  $LS$  cannot manipulate the reader to reuse an older  $t$  value. Thus, the adversary cannot link two locations to the same user.

Finally, the adversary controlling the RFID reader may attempt to reuse old  $t$  values to track a user. The adversary can program the RFID reader to always use the same time  $t$  value. The idea here is to try to get an RFID tag to return a response that has been repeated before. This way, the adversary can determine that that same tag has move pass the reader twice. Here, we let  $\epsilon$  to contain an always changing value  $n$  which is dependent on an incrementing counter  $ct$ . Therefore, even if the same same  $t$ ,  $R_{ID}$  and ID values are used, the resulting  $\epsilon$  will not be the same.

## 5.6 Conclusion

As ubiquitous systems move towards real world deployments, privacy systems that do not rely on trusted servers will become increasing important. In this Chapter, we propose an initial step towards more robust alternatives by allowing some of the servers to be compromised. Our future work considers two extensions. The first is to allow users to delegate data access control to other users, and the second is to explore techniques to improve range query performance.

## **Chapter 6**

# **IBE-Lite: A Lightweight Identity Based Cryptography for Body Sensor Networks**

This chapter considers remote data extraction from a sensor. Like the previous chapter, here we have a sensor wanting to ensure that its data once extracted remains secure. While the sensor is more powerful than a tag, the sensor also has more information to store than an RFID tag. In this chapter, we use a network of wearable sensors, a body sensor network, to demonstrate IBE-Lite, our solution to let a sensor efficiently encrypt data using multiple keys.

The use of wireless sensors for health care monitoring creates new ways of providing quality health care. A diverse array of specialized sensors can be deployed to monitor people like an at-risk patient with a history of heart attacks, or a senior citizen living independently at home. By continuously collecting a patient's physiological data in an unobtrusive manner, these sensors can provide doctors with additional information for better medical diagnosis. A body sensor network,

or BSN, is an important component in this monitoring scheme. A BSN consists of sensors placed directly on a patient's body or woven into the patient's clothes. A BSN "travels" with the patient, allowing for uninterrupted monitoring when the patient is outside his home.

The fact that a BSN is "always on", continuously collecting data, creates additional security and privacy demands. A patient will rightly want to limit the access and scope of the collected data to different people. For the purposes of this chapter, we assume the patient wishes to control data access according to the date, time, and the identity of the person who will access the data. For example, a patient may want to limit a physical therapist's access to BSN data collected on January 1st between 9 and 10 am, and no other times. In practice, there could be even more access conditions such as restricting data to specific locations where the data was collected.

In this chapter, we focus on a BSN deployed for medical monitoring. The data collected by the BSN is either stored in the sensors themselves, on a home computer, or forwarded to publicly accessible website. We use the term *storage site* to refer to where the data is stored. There is a certificate authority (CA) that helps a patient store and regulate access to decryption keys. Examples of possible CA include a local police department, or VA (veteran's affairs) clinic. A patient will register with a CA ahead of time, authorizing the CA to release permissions under different conditions to the appropriate personnel. A doctor wanting to obtain the data will first contact the CA for the appropriate keys, and then obtain the needed information from the storage site. We consider an adversary that seeks unauthorized access to the patient's data. These adversaries include criminal elements like identity thieves, as well as "snoopy" elements like employers. The adversary in the latter case may have *partial* access to some of the data, but may try to learn more. Techniques of these adversaries are elaborated in the security analysis later in the chapter.

We can provide the necessary security protections by designing the BSN to encrypt data with

different keys. For instance, the data collected between 9 and 10 am will be encrypted with a different key from the data collected between noon and 1pm the same day. This way, the patient can assign the appropriate decryption key to different people to limit access to the information.

**Symmetric key encryption :** In symmetric key encryption, the same key is used to both encrypt and decrypt the data. So for a patient wearing a BSN that monitors a patient 24 hours a day for an entire month and only wants his primary doctor to access the information, will need to store  $24 * 30 * 1 = 720$  symmetric keys in the BSN, assuming that a different key is used every hour. If the patient wishes to control access to different people (other doctors, in-house caregiver, and so on), then more keys will have to be assigned to the BSN.

A problem occurs when the BSN or a single sensor from the BSN is stolen. When this happens, the adversary will be able to decrypt the data, since the same key is used for both encryption and decryption. One solution is to increase the number of encryption keys by letting each sensor use a different key to encrypt the data collected at the same time. For a BSN with 100 sensors, we will have  $100 * 24 * 30 * 1 = 72000$  keys in the example given above. This makes key management more complicated since the decrypting party may not know in advance which key used. For this reason, many conventional protocols such as SSL on the Internet use symmetric keys to encrypt data, but use public keys to encrypt the symmetric key before transmission.

**Conventional public key encryption :** In conventional public key encryption like RSA, two keys are used, an encryption key and a decryption key. Now, we can store the encryption key on the BSN, and the decryption key safely elsewhere at a trusted location like the CA. When BSN is compromised, the adversary will only learn the encryption key and cannot decrypt the data.

However, the computation cost of using regular public key cryptography is high. Since the sensors in a BSN are computationally weak, implementing such a solution is infeasible on existing



BSN hardware. Furthermore, once the secret key is revealed, all encrypted data is vulnerable. This poses a problem when temporary access to the BSN data is needed. For instance, an on duty doctor wants to access the BSN data. If there is only one public key used to encrypt all data, the doctor after learning the secret key, will be able to decrypt all data even when off duty. A defense is to store many public keys in the BSN, leading to similar key management problems as with symmetric key encryption.

**Identity based encryption (IBE) :** IBE is a form of asymmetric cryptography like RSA. However, unlike RSA which requires both public and private key to be generated together, IBE allows a public key to be generated from an arbitrary string [20]. The corresponding private key can be generated separately later. For example, the patient may instruct the CA to release the keys to any ER doctor. Each day, the patient's BSN will create a new public key using the string  $str = \{date \mid time \mid ER\}$ . The CA does not have to create the corresponding private key. When an ER doctor wants to obtain data for Jan 1<sup>st</sup> between 9 and 10 am, he will first authenticate himself to the CA. The CA will then create the decryption key using that same string  $str = \{date \mid time \mid ER\}$ . This key can only decrypt data collected on that date and time.

Storing the syntax in the BSN is secure even if the sensors are compromised due to the asymmetric property of IBE. An adversary with access to a BSN sensor and knowledge of the syntax can only create a public key which cannot decrypt any information. Only the CA (or the patient himself) can create the private key to decrypt the data. Key management is also simplified, since the CA can generate a particular secret key based on the syntax, for instance date and time, on demand.

We design protocols based on identity-based encryption, IBE, that provide security and privacy protections while allowing flexible access to stored data. While IBE has been actively stud-

ied and widely applied in cryptography research, conventional IBE primitives are computationally demanding and cannot be efficiently implemented on BSN sensors. We developed IBE-Lite, a lightweight IBE suitable for a BSN. Through a proof-of-concept implementation of IBE-Lite on commercially available sensors, our experimental results show that IBE-Lite gives reasonable performance when executed by resource-constrained sensors.

## 6.1 Related Work

The motivation behind BSNs is to place low cost sensors directly on the patient for health care monitoring, and several research prototypes have been developed [73, 125]. Our work differs from these in that we focus on the security issues in BSNs.

Identity based encryption (IBE) is a relatively new type of asymmetric key encryption [20, 30]. Work by [124] developed protocols using IBE on sensors used in large scale sensor networks. The sensors used in a BSN have to worn on the patient, and are likely to be smaller and weaker. Thus unlike [124], our work uses IBE that does not rely on bilinear pairings such as Weil or Tate pairings in our primitives. Practical public key encryption for sensors have been proposed by [112–114], but these research only focus on conventional public keys, and do not support the IBE properties mentioned in this chapter.

Work by [80] uses IBE in a medical setting to secure the communications in a hospital, and [74] considered a security architecture for BSN focusing on the problem of key exchange between a sensor and a base station. The main difference between these work and our research is that we focus on deploying IBE on resource constrained devices to achieve practical performance. A key distinction is that our work presents evaluation results based on actual hardware.

Work by [9] proposed a secure system for BSNs using symmetric keys. While symmetric key

schemes use less storage space per key and generate a smaller ciphertext, they do not have the asymmetric property of public keys schemes like RSA or IBE-Lite. Our research takes advantage of recent advances that allows us to perform faster computation than earlier sensor hardware platforms. Another paper by [10] uses the variability of a patient’s heart rate as a means of person authentication. This work complements our IBE-Lite encryption since the patient’s heart rate can be used as an input string to generate encryption keys. [105] also consider using IBE on BSN, but suffers from slow query performance when searching over a lot of ciphertext.

## 6.2 IBE-Lite Solution

The simple examples presented in the previous section rely on conventional IBE which cannot be efficiently executed by a sensor in a BSN. Instead, we introduce *IBE-Lite*, a lightweight IBE that retains the properties of conventional IBE and yet can be executed on a BSN sensor is needed. The two useful properties are the ability to use an arbitrary string to generate a public key, and the ability to generate a public key separately from the corresponding secret key. IBE-Lite is built upon Elliptic Curve Cryptography (ECC), a public key primitive suitable for BSN [71].

Secret key	$x$	160 bits
Public parameters	$(y, P, p, q, h(.))$	1120 bits

**Table 6.1:** Size of basic ECC primitives.

To setup ECC, we need to derive a secret key  $x$ , and public parameters  $(y, P, p, q, h(.))$ . Table 6.1 shows the size of these parameters in bits. For the rest of the chapter, we denote encrypting a message  $m$  using public key  $y$  as  $EccEncrypt(m, y)$ , and decryption of ciphertext  $c$  generated by the  $EccEncrypt$  using the secret key  $x$  is given as  $EccDecrypt(c, x)$ . Details for generating the

parameters as well as *EccEncrypt* and *EccDecrypt* are found in [62].

### 6.2.1 IBE-Lite

From the basic ECC primitives, we derive the following IBE-Lite primitives, *setup*, *keygen*, *encrypt* and *decrypt*.

The intuition behind using IBE-Lite is to let a sensor independently generate a public key on-the-fly using an arbitrary string. For example, a sensor collecting EKG readings on Monday 1 pm will first create a string  $str = (\text{monday}|1\text{ pm}|EKG)$ . Using this string, the sensor can derive a public key,  $y_{str}$  to encrypt the data and send it to the storage site. There is no corresponding secret key created. In fact, the sensor *cannot* create the secret key needed to decrypt the message.

When the CA wishes to release this information to a doctor, the CA will derive the corresponding secret key,  $x_{str}$ , by using the same string  $str = (\text{monday}|1\text{ pm}|EKG)$ . This secret key only allows the doctor to decrypt messages encrypted by a sensor using the same string. This simplifies the key management, since the CA can generate the secret key on-demand without keeping track of which keys were used to encrypt which data. The only requirement is that the string used to describe the event is the same. Our primitives are as follows.

**Setup:** The patient selects an elliptic curve  $E$  over  $GF(p)$ , where  $p$  is a big prime number. We also denote  $P$  as the base point of  $E$  and  $q$  as the order of  $P$ , where  $q$  is also a big prime. The patient then generates  $n$  secret keys  $x_1, \dots, x_n \in GF(q)$  to generate the master secret key.

$$X = (x_1, \dots, x_n). \quad (6.1)$$

The  $n$  public keys are then generated to make up the master public key.

$$Y = (y_1, \dots, y_n), \quad (6.2)$$

where  $y_i = x_i \cdot P$ ,  $1 \leq i < n$ . Finally, the patient selects a collision resistant one-way hash function  $h: \{0, 1\}^* \rightarrow \{0, 1\}^n$ . The parameters

$$\langle Y, P, p, q, h(\cdot) \rangle \quad (6.3)$$

are released as the system public parameters.

**Keygen**: To derive a secret key  $x_{str}$  corresponding to a public key generated by a string  $str$ , the patient executes  $\text{Keygen}(str) = x_{str}$ ,

$$x_{str} = \sum_{i=1}^n h_i(str) \cdot x_i, \quad (6.4)$$

where  $h_i(str)$  is the  $i$ -th bit of  $h(str)$ .

**Encrypt**: To encrypt a message  $m$  using a public key derived from string  $str$ , the sensor does  $\text{Encrypt}(m, str)$  to determine the ciphertext  $c$ . Alg. 9 shows the process. Note that Alg. 9 lines 1 and 2 need only be run once to derive the public key  $y_{str}$ .

---

**Algorithm 9**  $\text{Encrypt}(m, str)$

---

1: Determine string  $str$  using agreed upon syntax

2: Generate public key  $y_{str}$  where

$$y_{str} = \sum_{i=1}^n h_i(str) \cdot y_i$$

3: Execute  $\text{EccEncrypt}(m, y_{str})$  to obtain  $c$

---

**Decrypt**: The doctor executes  $\text{Decrypt}(c, x_{str})$  to obtain the original message  $m$  which was encrypted using a secret key derived from  $str$ . The process is shown in Alg. 10.

## 6.2.2 BSN Security Protocols

Here we describe the protocols built upon IBE-Lite. First is the initialization phase where the patient first uses the BSN. Next is the data collection phase which outlines how a sensor encrypts

---

**Algorithm 10**  $\text{Decrypt}(c, str)$ 


---

- 1: Requests permission from CA to obtain data described by  $str$
  - 2: CA runs  $\text{Keygen}(str)$  to derive  $x_{str}$
  - 3: Doctor executes  $\text{EccDecrypt}(c, x_{str})$  to obtain  $m$
- 

the collected data. This is followed by the data transfer phase which describes how a BSN transfers data to a storage site. Finally, the query phase which occurs when a doctor needs to obtain data from the storage site.

We assume that an agreed upon syntax is used to describe the public key, and this description is termed as  $str$ . For example, the patient deciding to collect data on a hourly basis will set the sensors in the BSN to affix a timestamp rounded to the nearest hour when creating  $str$ . In other words, two EKG readings collected on Monday at 1:05 pm and 1:20 pm will both be described using the same string  $str = \{monday|1\ pm|EKG\}$ .

As mentioned earlier, we assume an honest-but-curious storage site which will try to learn the contents of the stored data, but will otherwise not delete the stored data. We also assume a separate security mechanism is in place so that only the patient can store BSN data onto the storage site.

**Initialization:** The patient first executes **Setup** to obtain the master secret key  $X = (x_1, \dots, x_n)$ , and public parameters  $\langle Y, P, p, q, h(\cdot) \rangle$ . The patient loads the parameters  $\langle Y, P, p, q, h(\cdot) \rangle$  into every sensor in the BSN. The master secret key is registered with the CA.

**Data collection:** Let the sensor collect data  $d$  at event  $str$ . The sensor executes Alg. 11 to encrypt its data. The tuple  $(c_1, c_2)$  is then stored in sensor memory. The  $flag$  is a commonly known bitstring several bits long.

**Data transfer:** Periodically, each sensor in the BSN will transfer its data to the storage site. This is done by first aggregating all the data into a cellphone like device [125]. The cellphone

---

**Algorithm 11** Sensor encrypting data
 

---

- 1: Derive the string  $str$ , and generate a random number  $n$
  - 2: Calculate  $m_1 = (flag|n)$  where  $flag$  is a known bitstring
  - 3: Calculate  $m_2 = (d|n)$
  - 4: Calculate  $c_1 = \text{Encrypt}(str, m_1)$
  - 5: Calculate  $c_2 = \text{Encrypt}(str, m_2)$
- 

then forwards the aggregated data to the storage site. Assuming that there are  $k$  tuples generated by the BSN, the cellphone will forward the set  $\{(c_1^1, c_2^1), \dots, (c_1^k, c_2^k)\}$ . Alternatively, a sensor with enough storage capacity can opt to store the data within the sensor itself. In this case, there is no data transfer process.

**Querying:** A doctor wishing to obtain data collected under some  $str$  will first contact the CA for permission. After the CA agrees, the CA will run  $\text{Keygen}(str)$  to derive the corresponding secret key  $x_{str}$  needed to decrypt data.

The doctor then contacts the storage site and retrieves the data as shown in Alg. 12. When the data is stored within the sensor, the role of the storage site will be executed by the sensor.

Since all the data is encrypted, the storage site cannot return a specific encrypted data to the doctor. Instead, the storage site simply lets the doctor try to decrypt each tuple  $(c_1, c_2)$  belonging to the patient. The reason the storage site first returns  $c_1$  for the doctor to verify instead of returning  $c_2$  directly is to improve efficiency. Since the length of  $c_1$  is much shorter than that of  $c_2$ , letting the doctor first attempt to decrypt  $c_1$  before sending the much longer  $c_2$  reduces transmission time.

Notice that  $c_2$  embeds the same random number  $n$  in both  $c_1$  and  $c_2$ , and the doctor will only accept the data in  $c_2$  to be legitimate only if both random numbers match. This random  $n$  is known only to the sensor encrypting the data. Consider for example two sensors belonging to the same

---

**Algorithm 12** Doctor querying for data
 

---

```

1: for every  $(c_1^i, c_2^i) \ i \in k$  for patient do
2:   Storage site sends  $c_1^i$  to doctor
3:   Doctor runs  $\text{Decrypt}(c_1^i, str)$ 
4:   if the initial bits of the result match flag then
5:     Doctor requests corresponding  $c_2^i$  from storage site
6:     Doctor executes  $\text{Decrypt}(c_2^i, str)$  and checks whether the  $n$  matches the value from  $c_1^i$ 
7:     Doctor accepts  $d$  if both are correct
8:   end if
9: end for

```

---

BSN encrypting some data using the same string  $str$ . Since both sensors are legitimate, the use of the random  $n$  prevents an adversary from swapping the  $c_2$ s from different sensors to confuse the doctor.

### 6.2.3 Query Improvements

A potential bottleneck is the amount of time needed for a doctor to query a storage site. Consider a storage site with  $k$  tuples  $(c_1^1, c_2^1), \dots, (c_1^k, c_2^k)$ , and a doctor receives  $l$  secret keys from the patient. The storage site will have to transmit  $c_1^i, i \in k$  to the doctor, and the doctor will have to try every key  $x_j, j \in l$  on each  $c_1^i$  to determine whether there is any desired data in the storage site. This takes  $O(k \cdot l)$  amount of time.

This poor performance is because the storage site is unable to index any of the tuples since the storage site cannot determine the actual content of the tuples. This feature protects the privacy of the patient at the cost of slower searching time. For instance, consider a storage site have



many tuples belonging to the same patient, and one of the tuples is encrypted using the string  $str = \{date \mid ER\}$ . An ER doctor with the corresponding secret key will still have to go through every tuple in the storage site to determine whether that single tuple. This is inefficient when a storage site contains many different tuples.

We can improve the search performance by letting the sensor encrypt additional hints about the tuples. This hint is a variable that can summarize several tuples together. For example, the sensor may have created two tuples  $(c_1^1, c_2^1)$  and  $(c_1^2, c_2^2)$  using two different descriptions  $\{date \mid ER\}$  and  $\{date \mid gym\}$ . Since both descriptions contain the same condition  $date$ , we can create a hint  $\eta = \text{Encrypt}(m, str)$  where

$$m = (flag \mid n \mid i_1^1 \mid i_1^2) \quad (6.5)$$

and  $str = \{date\}$ . Here  $i_1^1$  and  $i_1^2$  refer to the indices pointing to  $c_1^1$  and  $c_1^2$ .

Now the doctor requesting permissions will get an extra key from the CA for the date to decrypt the hint. The use of hints improve the performance by reducing the number of transmissions between the storage site and the doctor, since the doctor will only request  $c_1$ s from hints he can decrypt. This scheme is still secure since the doctor still needs the correct key  $x_{str}$  to decrypt a particular  $c_1$ . The privacy of the patient is still protected from the storage site since the storage site learns nothing from the hints.

## 6.3 Analysis and Evaluation

### 6.3.1 Security

Here, we analyze the security of our proposed protocols. Encryption and decryption are performed using the keys  $x_{str}$  and  $y_{str}$  derived from string  $str$ . Both  $x_{str}$  and  $y_{str}$  do not violate the discrete

logarithm property where, given  $y = x \cdot P$ , it is infeasible to determine  $x$  given  $y$  and  $P$ , since both are derived from addition of points on the same curve.

**Eavesdropping attack:** In this attack, the adversary eavesdrop on the message transmitted from the BSN to the storage site and learns the tuple  $(c_1, c_2)$ . The adversary succeeds in his attack if he is able to determine the data  $d$  after observing as many tuples as he wishes. Since our protocol encrypts all data before broadcast, the adversary learns nothing from the ciphertext.

**Tracking attack:** Here, the adversary attacks the patient's privacy by observing multiple transmission between a BSN and a storage site. The adversary is considered able to track the patient if given two tuples, the adversary is able to determine whether they come from the same BSN. In our protocol, each ciphertext  $(c_1, c_2)$  includes a new random number  $n$ . In fact, even if *identical* data encrypted using a public key derived from the same string  $str$  in two different broadcasts cannot be linked together since a different random  $n$  will be used. This is important when the BSN monitors data such as body temperature which may remain relatively static for long periods of time.

**Compromised sensor:** We assume that the adversary compromises one or more sensors in the BSN, and is able to extract all data that is stored on the sensor. The adversary succeeds in this attack if he is able to use the information to determine previously encrypted data. Since each sensor only stores the public parameters  $(Y, P, p, q, h(\cdot))$ , the adversary learns no secret knowledge which can enable him to decrypt any tuples  $(c_1, c_2)$ .

**Matching attack:** The adversary launches a matching attack by first creating many public keys using different strings  $str$ . The adversary then encrypts all possible values using the different public keys to determine whether there is a match for the tuple  $(c_1, c_2)$ . This is possible since the number of potential EKG readings for example are bounded. However, both  $c_1$  and  $c_2$  contains a

random number  $n$  generated by the sensor. Since the adversary cannot predict the value of  $n$ , the matching attack fails.

**Honest-but-curious storage site:** This type of storage site will not delete the user's data but may attempt to determine the contents of the data. This assumption is common for many web based applications. For instance, an email service provider can generally be assumed to *not* delete the user's emails, but may try to use some of the content to place advertisements. This requirement also covers instances where the storage site is compromised and data exposed to an adversary. In our protocol, all data stored on the storage site is encrypted, and no secret keys are stored in the storage site. Therefore, an adversary with access to all the ciphertexts cannot decrypt the data.

Note that an malicious storage site can still cause disruption by deleting the patient's data. While our protocols do not prevent this, a practical defense is to store the same encrypted data at different storage sites so that the data is still recoverable.

**Complexity analysis:** Given  $n$  public keys  $Y = (y_1, \dots, y_n)$ , the time complexity for using IBE-Lite to generate a public key  $y_{str}$  using string  $str$  is  $O(n)$ , and the time needed to perform the encryption with  $y_{str}$  is  $O(1)$ . Similarly, the decryption requires a time complexity of  $O(n)$  to generate the secret key  $x_{str}$  and  $O(1)$  to decrypt the data.

Since our protocols rely on asymmetric key encryption, only public keys are stored in the sensors. Thus, our schemes are resistant against an adversary that can compromise all sensors in the BSN. However, our protocols are vulnerable to attack if there are  $O(n)$  colluding users each with a single secret key  $x_{str}$ . The colluding users can use their individual secret keys to derive the master secret key  $X$  given in equation 1. This vulnerability can be defended by the rekeying process given below.

**Rekeying:** A vulnerability of our scheme is that once  $n$  secret keys  $x_{str}^1, \dots, x_{str}^n$ , are released,

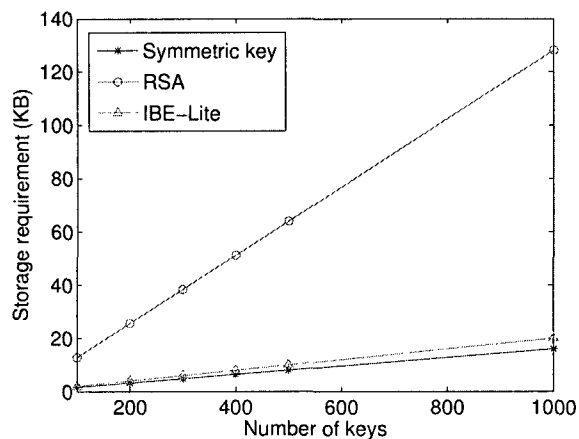
colluding users will be able to solve for the master secret  $X$ . There are two defenses against this. First, we can select a large enough  $n$  such that we will never release  $n$  secret keys. As we will show in later, we can easily let  $n$  be a few thousand keys without incurring heavy storage penalty. Note the large value of  $n$  only refers the number of secret keys *released*, and not the amount of data *encrypted*. For instance, for  $n = 500$ , the BSN can still encrypt more than 500 pieces of data so long as less than 500 secret keys are released. This is unlike a symmetric key or conventional public key solution where each piece of encrypted data requires a new key. This distinction is important because a BSN that is continuously worn by a patient will always collect information, but only a subset may be ever be used. Since we cannot determine in advance what data will be requested, we therefore have to ensure we have enough keys to encrypt everything.

Second, we can rekey the BSN by creating a new set of  $n$  secret keys as the master secret, and store the new public information in the sensors. Note that the rekeying does **not** have to be done by the BSA itself. A powerful laptop can be used, and the information then stored into the BSN. The laptop can then re-register the new keys with the CA securely online. This is the same as changing a password for a bank account. The patient does not have to physically visit the CA. To reduce the rekeying frequency, the CA can be configured to inform the patient when to rekey his BSN after it has released a certain number of secret keys. This way, if the data collected by the BSN has not been requested by any doctor, the patient can avoid the overhead of rekeying.

### 6.3.2 Performance

We evaluate our protocols using experiments conducted on commercially available Tmote Sky sensors from MoteIV. The Tmote Sky is the next-generate sensor hardware module for extremely low power, cost effective, and reliable sensor network applications. Tmote Sky has a 8MHz TI

MSP430 CPU, 10KB on-chip RAM, 48KB programming ROM. Equipped with 802.15.4/ZigBee radio, an integrated antenna provides up to 125 meter radio transmission range.

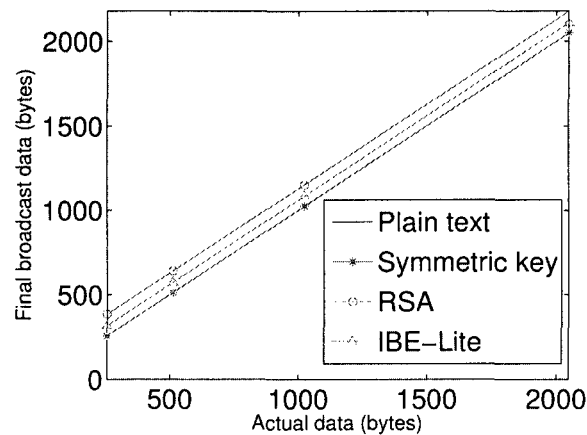


**Figure 6.1:** Amount of storage needed to store  $n$  keys for different encryption methods.

Fig. 6.1 shows the amount of storage needed for different encryption schemes. We represent a conventional asymmetric key encryption scheme using RSA. Note that we also need to store many encryption keys for both symmetric key and RSA encryption schemes. We see that a symmetric key encryption requires the least amount of storage, while RSA encryption uses the most amount of storage. Our IBE-Lite gives us the advantages of asymmetric key encryption while using a little more storage space than a symmetric key scheme.

Fig. 6.2 shows the data transmission overhead of the various encryption schemes. For both the RSA and IBE-Lite, the data itself is not encrypted. Instead, a symmetric key is first used to encrypt the data, and then the asymmetric key is used to encrypt the data. This is the conventional method when designing protocols using asymmetric key encryption.

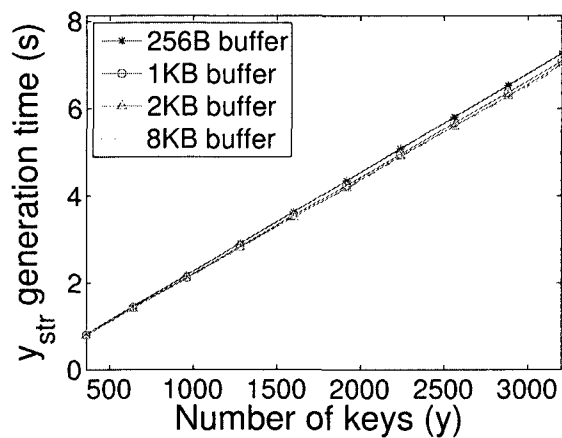
The main overhead of IBE-Lite over other encryption schemes is the time needed to generate an encryption key  $y_{str}$  from a string  $str$  using  $n$  number of public keys  $y_1, \dots, y_n$ . In both symmetric key and RSA, the public keys are precomputed and stored in the sensor. Fig. 6.3 shows the amount



**Figure 6.2:** Data transmission overhead for different encryption schemes. All values in bytes.

of time needed to generate a single  $y_{str}$  with varying values of  $n$ . All  $n$  public keys are initially stored in the flash memory. We see for instance that for  $n = 360$ , we need only 0.9 seconds to generate  $y_{str}$ . While IBE-Lite does require an additional key generation time, we note that we can achieve the properties of asymmetric key encryption like RSA using a much small amount of storage space. In addition, the amount of time a single public key can be used is typically much longer than the time needed to generated a new key.

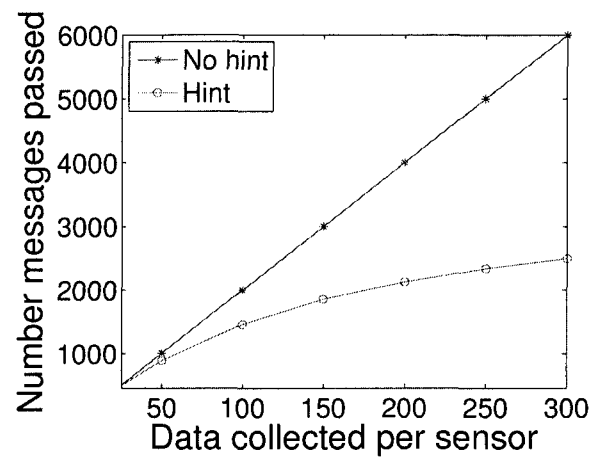
We use simulation to evaluate the search improvements. Since the searching is performed by a doctor with access to a more powerful machine, the simulations focused on the number of messages passed between a doctor and a storage site. We assume there are 1000 different possible time periods where a sensor may collect data. A sensor randomly selects a time period to collect data, and encrypts the data using a public key derived from that time period. The doctor is assumed to randomly select data from five time periods. The results are the average over 100 trials. Fig. 6.4 shows the improvement when hints are used.



**Figure 6.3:** Time needed to derive one  $y_{str}$  using different  $n$  number of public keys,  $y_1, \dots, y_n$ .

## 6.4 Conclusion

In this chapter, we presented IBE-Lite, a lightweight identity based encryption method suitable for a body sensor network. We provided protocols based on IBE-Lite and evaluated the protocols using a combination of security analysis, simulations, and practical implementation on actual sensors.



**Figure 6.4:** Comparing the difference in number of message passed during search when using hints verses no hints.



## Chapter 7

# Conclusion and Future Work

Ubiquitous computing uses small, networked computing devices embedded into physical objects to create different applications. This dissertation focuses on the problem of how to securely and efficiently extract information from these devices. We organize these devices into two categories, battery-free RFID tags, and battery-powered sensor motes. For each device category, we consider two types of data access, a local and a remote access. In a local access, the tag or sensor only needs to authenticate the application querying for information, but in a remote access, the small device needs to perform additional processing to ensure that the collected data remains secure.

RFID tags are inexpensive devices that can only perform very limited operations because they draw their operating power from the RFID reader querying the tags. We make two main contributions for local RFID tag data access. In our first contribution, we introduced authentication and search protocols that allow an RFID reader to securely extract data from a single RFID tag. Our protocols allow the RFID reader to independently access the RFID tag without requiring access to the backend server. In our second contribution, we designed protocols to accurately and efficiently monitor a set of RFID tags for missing tags without actually collecting all the tag IDs. The protocols can be applied even if the person performing the monitoring is malicious.

For remote RFID tag data access, we designed a protocol for RFID based object tracking system that divides a tag reply into two databases. Our protocol protects privacy even when one of the databases is controlled by an adversary.

Sensors are equipped with a (weak) processor, some sensing capabilities, and can perform more complex operations than RFID tags. However, unlike RFID tags, sensors are limited by their battery power. Our main contribution for local sensor data access lies in our design and implementation of a search system akin to a desktop search engine that indexes the information inside a small device and accurately resolves a users queries. We modified information retrieval (IR) techniques and proposed a memory efficient top-k query resolution algorithm that returns accurate answers while utilizing limited amounts of memory. Our main contribution for remote sensor data access is a lightweight identity-based encryption technique that allows a sensor to generate asymmetric cryptographic keys based on human-readable strings. This allows a sensor to easily generate sufficient keys to encrypt each piece of data with a different key.

## 7.1 Future work

There are two natural questions that arise from secure and efficient techniques to extract information from small devices. The two questions are,

1. How do we determine the correctness of the information collected by these small devices?
2. How do we manage all the information collected by these small devices?

The first question is an issue of information assurance for the data collected by these small devices. While our current research allows us to securely extract data from this sensor, we are unable to determine the **correctness** of the data, that is whether the collected data is an accurate

representation of reality. To illustrate, consider a glucose sensor that is attached to a person's body and used to continuously monitor his glucose levels. Examples of problems that will affect the correctness of the collected data are

- the position where the sensor is placed,
- whether sensor has been calibrated correctly,
- and the identity of the person the sensor is placed on.

We have made some preliminary progress on the first problem. We attempt to determine a sensor's position with *centimeter-range accuracy* using a combination of both sensors and RFID tags. This is a departure from existing wireless localization solutions that use signal strength to determine positioning and are only accurate up to several meters. The intuition behind our solution is to have each sensor be equipped with a miniature RFID reader. Once the sensor is placed on the user's body, the sensor will read the IDs of the surrounding RFID tags that are woven into the user's clothing. This information is then used to form a location proof to verify that the sensor is placed correctly. Our proposed solution is supported by existing sensor and RFID hardware. Small sensors such as the Skyetek M1-Mini have a built in RFID reader that can be used to read the RFID tags, and 13.56 MHz RFID tags with a short communication range of approximately five centimeters can be used for positioning. The remaining challenge is to ensure that the interaction between the sensor and the RFID tags is resistant to adversary attacks.

The second question is to consider techniques to manage the information **after** it has been extracted. Ubiquitous computing surrounds us with many small devices that are continuously collecting data. The amount of data collected can be very large, and will eventually have to be stored at a reliable facility. One such facility is a cloud computing service provider which will

provide reliable access for other users and perform other important tasks like maintaining regular backups. However, this requires the user to trust the service provider with all his data. The research challenge in this direction is to balance the efficiency cloud computing services provides with the user's control over the security of his data. Some specific problems to be considered are

- efficient encrypted data retrieval,
- data management operations (top-k, most frequent, and so on) on encrypted data,
- and techniques to verify the integrity of the stored data.

Our existing research on remote access only considers these problems from the standpoint of the tag or sensor, but does not consider what modifications can be performed by the cloud service provider to improve performance. We intend to expend our research to consider the role of the cloud service provider in addition to that of the small to solve the problem of secure and efficient data access.

Finally, during the course of writing this dissertation, we have seen some hybrid devices such as solar powered sensors and programmable RFID tags enter the market. The nature of these hybrid devices challenge our division of small devices into simple battery-free RFID tags, and more powerful battery powered sensors. We believe improvements in hardware technologies will continue to introduce new types of small devices with different characteristics that will continue to be a source of interesting research problems.

# Bibliography

- [1] A CHRONOLOGY OF DATA BREACHES. <http://www.privacyrights.org/ar/chrondatabreaches.htm>, 2009.
- [2] GREGORY D. ABOWD, CHRISTOPHER G. ATKESON, JASON HONG, SUE LONG, ROB KOOPER, AND MIKE PINKERTON. Cyberguide: a mobile context-aware tour guide. *Wirel. Netw.*, 3(5):421–433, 1997.
- [3] RAKESH AGRAWAL, JERRY KIERNAN, RAMAKRISHNAN SRIKANT, AND YIRONG XU. Hippocratic databases. In *VLDB*, 2002.
- [4] DENISE ANTHONY, TRISTAN HENDERSON, AND DAVID KOTZ. Privacy in location-aware computing environments. *IEEE Pervasive Computing*, 2007.
- [5] APPLE. <http://www.apple.com/macosx/features/spotlight/>.
- [6] GILDAS AVOINE. <http://lasecwww.epfl.ch/~gavoine/rfid/>.
- [7] GILDAS AVOINE AND PHILIPPE OECHSLIN. A Scalable and Provably Secure Hash Based RFID Protocol. In *International Workshop on Pervasive Computing and Communication Security (PerSec)*, 2005.
- [8] RICARDO BAEZA-YATES, GEORGES DUPRET, AND HAVIER VELASCO. A study of mobile search queries in japan. In *Proceedings of the International World Wide Web Conference*, 2006.
- [9] SHU-DI BAO, YUAN-TING ZHANG, AND LIAN-FENG SHEN. A new symmetric cryptosystem of body area sensor networks for telemedicine. In *Proceedings of the Conference the Japan Society of Medical Electronics and Biological Engineering*, 2005.
- [10] SHU-DI BAO, YUAN-TING ZHANG, AND LIAN-FENG SHEN. Physiological signal based entity authentication for body area sensor networks and mobile healthcare systems. In *IEEE Engineering in Medicine and Biology*, 2005.
- [11] LEJLA BATINA, JORGE GUAJARDO, TIM KERINS, NELE MENTENS, PIM TUYLES, AND INGRID VERBAUWHEDE. An elliptic curve processor suitable for RFID-tags. Cryptology ePrint Archive, Report 2006/227.
- [12] LEJLA BATINA, JORGE GUAJARDO, TIM KERINS, NELE MENTENS, PIM TUYLES, AND INGRID VERBAUWHEDE. Public key cryptography for RFID-tags. RFIDSec 06.
- [13] BEAGLE. [http://beagle-project.org/main\\_page](http://beagle-project.org/main_page).
- [14] ALASTAIR R. BERESFORD AND FRANK STAJANO. Mix zones: User privacy in location-aware services. In *Pervasive Computing and Communications Workshops (PERCOMW)*, 2004.

- [15] A.R. BERESFORD AND F. STAJANO. Location privacy in pervasive computing. *IEEE Pervasive Computing*, 2003.
- [16] ELISA BERTINO AND RAVI SANDHU. Database security-concepts, approaches, and challenges. *IEEE Transactions on Dependable and Secure Computing*, 2005.
- [17] LEONID BOLOTNYY AND GABRIEL ROBINS. Generalized “Yoking-Proofs” for a group of RFID tags. In *Mobiquitous*, 2006.
- [18] LEONID BOLOTNYY AND GABRIEL ROBINS. Physically unclonable function -based security and privacy in rfid systems. In *International Conference on Pervasive Computing and Communications (PerCom)*, 2007.
- [19] DAN BONEH, GIOVANNI DI CRESCENZO, RAFAIL OSTROVSKY, AND GIUSEPPE PERSIANO. Public key encryption with keyword search. In *EUROCRYPT*, 2004.
- [20] DAN BONEH AND MATT FRANKLIN. Identity-based encryption from the Weil pairing. In *CRYPTO*, 2001.
- [21] MAURIZIO A. BONUCCELLI, FRANCESCA LONETTI, AND FRANCESCA MARTELLI. Tree slotted ALOHA: a new protocol for tag identification in RFID networks. In *WoWMoM*, 2006.
- [22] JULIEN BRINGER, HERVÉ CHABANNE, AND DOTTAX EMMANUELLE. HB<sup>++</sup>: a lightweight authentication protocol secure against some attacks. In *SecPerU*, 2006.
- [23] CLAUDE CASTELLUCCIA AND GILDAS AVOINE. Noisy Tags: A Pretty Good Key Exchange Protocol for RFID Tags. In *International Conference on Smart Card Research and Advanced Applications (CARDIS)*, 2006.
- [24] JAE-RYONG CHA AND JAE-HYUN KIM. Novel anti-collision algorithms for fast object identification in RFID system. In *ICPADS*, 2005.
- [25] CHRISTY CHATMON, TRI VAN LE, AND MIKE BURMESTER. Secure anonymous RFID authentication protocols. Technical report, Florida State University, Department of Computer Science, 2006.
- [26] JIANGPING CHEN, ANNE DIEKEMA, MARY D. TAFFET, NANCY J. MCCRACKEN, NECATI ER-CAN OZGENCIL, OZGUR YILMAZEL, AND ELIZABETH D. LIDDY. Question answering: CNLP at the TREC-10 question answering track. In *Text REtrieval Conference*, 2001.
- [27] KEITH CHEVERST, NIGEL DAVIES, KEITH MITCHELL, AND ADRIAN FRIDAY. Experiences of developing and deploying a context-aware tourist guide: the guide project. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 20–31, New York, NY, USA, 2000. ACM.
- [28] KEITH CHEVERST, NIGEL DAVIES, KEITH MITCHELL, ADRIAN FRIDAY, AND CHRISTOS EPSTRATIOU. Developing a context-aware electronic tourist guide: some issues and experiences. In *CHI '00: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 17–24, New York, NY, USA, 2000. ACM.
- [29] KAREN CHURCH, BARRY SMYTH, PAUL COTTER, AND KEITH BRADLEY. Mobile information access: A study of emerging search behavior on the mobile internet. *ACM Trans. Web*, 1(1):4, 2007.
- [30] C. COCKS. An identity based encryption scheme based on. quadratic residues. In *LNCS 2260*, 2001.
- [31] COMMERCE DEPARTMENT: WE LOSE LAPTOPS. <http://arstechnica.com/security/news/2006/09/7809.ars>, 2006.

- [32] ALEPH1 COMPANY. Yaffs: Yet another flash file system. In <http://www.yaffs.net/>, 2008.
- [33] HUI DAI, MICHAEL NEUFELD, AND RICHARD HAN. Elf: an efficient log-structured flash file system for micro sensor nodes. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 176–187, New York, NY, USA, 2004. ACM.
- [34] TASSOS DIMITRIOU. A lightweight RFID protocol to protect against traceability and cloning attacks. In *SecureComm*, 2005.
- [35] ROGER DINGLEDINE, NICK MATHEWSON, AND PAUL SYVERSON. Tor: the second-generation onion router. In *USENIX Security Symposium*, 2004.
- [36] CHRIS FALOUTSOS. Access methods for text. *ACM Comput. Surv.*, 17(1), 1985.
- [37] CHRISTOS FALOUTSOS AND DOUGLAS W. OARD. A survey of information retrieval and filtering methods. Technical Report CS-TR-3514, University of Maryland at College Park, 1995.
- [38] FBI LOST 160 LAPTOPS IN LAST 44 MONTHS. <http://arstechnica.com/old/content/2007/02/8821.ars>, 2007.
- [39] MARTIN FELDHOFFER AND CHRISTIAN RECHBERGER. A case against currently used hash functions in rfid protocols. In *OTM Workshops (1)*, 2006.
- [40] William B. Frakes and Ricardo A. Baeza-Yates, editors. *Information Retrieval: Data Structures and Algorithms*. Prentice-Hall, 1992.
- [41] JAMES C. FRENCH, ALLISON L. POWELL, JAMES P. CALLAN, CHARLES L. VILES, TRAVIS EMMITT, KEVIN J. PREY, AND YUN MOU. Comparing the performance of database selection algorithms. In *Research and Development in Information Retrieval*, 1999.
- [42] ERAN GAL AND SIVAN TOLEDO. Algorithms and data structures for flash memories. *ACM Comput. Surv.*, 37(2), 2005.
- [43] ERAN GAL AND SIVAN TOLEDO. A transactional flash file system for microcontrollers. In *ATEC '05: Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 7–7, Berkeley, CA, USA, 2005. USENIX Association.
- [44] HENRI GILBERT, MATTHEW ROBshaw, AND HERVÉ SIBERT. An active attack against HB<sup>+</sup> – a provably secure lightweight authentication protocol. Manuscript, 2005.
- [45] HECTOR GONZALEZ, JIAWEI HAN, AND XIAOLEI LI. Mining compressed commodity workflows from massive rfid data sets. In *CIKM*, 2006.
- [46] HECTOR GONZALEZ, JIAWEI HAN, XIAOLEI LI, AND DIEGO KLABJAN. Warehousing and analyzing massive rfid data sets. In *ICDE*, 2006.
- [47] GOOGLE. [www.desktop.google.com](http://www.desktop.google.com).
- [48] D. HAHNEL, W. BURGARD, D. FOX, K. FISHKIN, AND M. PHILIPOSE. Mapping and localization with rfid technology. In *IEEE International Conference on Robotics and Automation*, 2004.
- [49] URS HENGARTNER AND PETER STEENKISTE. Access control to people location information. *ACM Trans. Inf. Syst. Secur.*, 2005.
- [50] A. HERZBERG, H. KRAWCZYK, AND G. TSUDI. On travelling incognito. In *IEEE Workshop on Mobile Computing Systems and Applications*, 1994.

- [51] RICHARD HOLLINGER AND JASON DAVIS. National retail security survey 2001.
- [52] JASON I. HONG AND JAMES A. LANDAY. An architecture for privacy-sensitive ubiquitous computing. In *International conference on Mobile systems, applications, and services (MobiSys)*, 2004.
- [53] INTEL. [www.intel.com/research/downloads/imote-ds-101.pdf](http://www.intel.com/research/downloads/imote-ds-101.pdf).
- [54] ARI JUELS. “Yoking-Proofs” for RFID tags. In *Pervasive Computing and Communications Workshops*, 2004.
- [55] ARI JUELS. RFID security and privacy: A research survey. Manuscript, 2005.
- [56] ARI JUELS. Strengthening EPC tags against cloning. In *WiSe*, 2005.
- [57] ARI JUELS AND RAVIKANTH PAPPU. Squealing euros: Privacy protection in RFID-enabled banknotes. In *Financial Cryptography*, 2003.
- [58] ARI JUELS AND STEPHEN WEIS. Authenticating pervasive devices with human protocols. In *Advances in Cryptology – CRYPTO’05*, 2005.
- [59] MARYAM KAMVAR AND SHUMEET BALUJA. A large scale study of wireless search behavior: Google mobile search. In *CHI ’06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 701–709, New York, NY, USA, 2006. ACM.
- [60] APU KAPADIA, TRISTAN HENDERSON, JEFFREY J. FIELDING, AND DAVID KOTZ. Virtual walls: Protecting digital privacy in pervasive environments. In *Proceedings of the Fifth International Conference on Pervasive Computing (Pervasive)*, 2007.
- [61] MEI KOBAYASHI AND KOICHI TAKEDA. Information retrieval on the web. *ACM Comput. Surv.*, 32(2):144–173, 2000.
- [62] N. KOBLITZ. Elliptic curve cryptosystems. In *Mathematics of Computation*, Vol.48, 1987.
- [63] MURALI KODIALAM AND THYAGA NANDAGOPAL. Fast and reliable estimation schemes in RFID systems. In *MobiCom*, 2006.
- [64] TRAVIS KRIPLEAN, EVAN WELBOURNE, NODIRA KHOUSSAINOVA, VIBHOR RASTOGI, MAGDALENA BALAZINSKA, GAETANO BORRIELLO, TADAYOSHI KOHNO, AND DAN SUCIU. Physical access control for captured rfid data. *IEEE Pervasive Computing*, 2007.
- [65] SANDEEP KUMAR AND CHRISTOF PAAR. Are standards compliant elliptic curve cryptosystems feasible on RFID? RFIDSec 06.
- [66] S. LEDERER, J. I. HONG, X. JIANG, A. K. DEY, J. A. LANDAY, AND J. MANKOFF. Towards everyday privacy for ubiquitous computing. Technical Report UCB-CSD-03-1283, Computer Science Division, University of California, Berkeley, 2003.
- [67] SU-MI LEE, YOUNG JU HWANG, DONG HOON LEE, AND JONG IN LIM LIM. Efficient authentication for low-cost RFID systems. In *ICCSA 2005*, 2005.
- [68] SU-RYUN LEE, SUNG-DON JOO, AND CHAE-WOO LEE. An enhanced dynamic framed slotted ALOHA algorithm for RFID tag identification. In *Mobiquitous*, 2005.
- [69] TIEYAN LI AND ROBERT H. DENG. Vulnerability analysis of EMAP - an efficient RFID mutual authentication protocol. In *Second International Conference on Availability, Reliability and Security*, 2007.



- [70] TIEYAN LI AND GUILIN WANG. Security analysis of two ultra-lightweight RFID authentication protocols. In *IFIP SEC*, 2007.
- [71] BENNY LO AND GUANG ZHONG YANG. Key technical challenges and current implementations of body sensor networks. In *BSN*, 2005.
- [72] LOGITEC. [www.logitech.com](http://www.logitech.com).
- [73] DAVID MALAN, THADDEUS FULFORD-JONES, MATT WELSH, AND STEVE MOULTON. Codeblue: An ad hoc sensor network infrastructure for emergency medical care. In *BSN*, 2004.
- [74] KRIANGSIRI MALASRI AND LAN WANG. Addressing security in medical sensor networks. In *HealthNet*, 2007.
- [75] GAURAV MATHUR, PETER DESNOYERS, DEEPAK GANESAN, AND PRASHANT SHENOY. Capsule: an energy-optimized object storage system for memory-constrained sensor devices. In *SenSys '06: Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 195–208, New York, NY, USA, 2006. ACM.
- [76] GAURAV MATHUR, PETER DESNOYERS, DEEPAK GANESAN, AND PRASHANT SHENOY. Ultra-low power data storage for sensor networks. In *IPSN '06: Proceedings of the 5th international conference on Information processing in sensor networks*, pages 374–381, New York, NY, USA, 2006. ACM.
- [77] A. MICIC, A. NAYAK, D. SIMPLOT-RYL, AND I. STOJMENOVIC. A hybrid randomized protocol for RFID tag identification. In *WoNGeN*, 2005.
- [78] DAVID MOLNAR AND DAVID WAGNER. Privacy and security in library RFID: Issues, practices, and architectures. In *CCS*, 2004.
- [79] DAVID MOLNAR AND DAVID WAGNER. Privacy and Security in Library RFID: Issues, Practices, and Architectures. In *Conference on Computer and Communications Security*, 2004.
- [80] M.C. MONT, P BRAMHALL, AND K. HARRISON. A flexible role-based secure messaging service: exploiting IBE technology for privacy in health care. In *International Workshop on Database and Expert Systems Applications*, 2003.
- [81] GINGER MYLES, ADRIAN FRIDAY, AND NIGEL DAVIES. Preserving privacy in environments with location-based applications. *IEEE Pervasive Computing*, 2003.
- [82] MIYAKO OHKUBO, KOUTAROU SUZUKI, AND SHINGO KINOSHITA. Cryptographic approach to “privacy-friendly” tags. In *RFID Privacy Workshop*, 2003.
- [83] KHALED OUAFI AND RAPHAEL C.-W. PHAN. Privacy of Recent RFID Authentication Protocols. In *4th International Conference on Information Security Practice and Experience – ISPEC 2008*, 2008.
- [84] PEDRO PERIS-LOPEZ, JULIO C. HERNANDEZ-CASTRO, JUAN M. ESTEVEZ-TAPIADOR, AND ARTURO RIBAGORDA. Solving the simultaneous scanning problem anonymously: Clumping proofs for RFID tags. In *SecPerU*, 2007.
- [85] ROBERTO DI PIETRO AND REFIK MOLVA. Information Confinement, Privacy, and Security in RFID Systems. In *European Symposium On Research In Computer Security (ESORICS)*, 2007.
- [86] SELWYN PIRAMUTHU. HB and related lightweight authentication protocols for secure RFID tag/reader authentication. In *COLLECTeR*, 2006.

- [87] SELWYN PIRAMUTHU. On existence proofs for multiple RFID tags. In *ICPS*, 2006.
- [88] PHILIPPE PUCHERAL, LUC BOUGANIM, PATRICK VALDURIEZ, AND CHRISTOPHE BOBINEAU. Picodbms: Scaling down database techniques for the smartcard. *The VLDB Journal*, 10(2-3):120–132, 2001.
- [89] CHANDRAMOULI RAMASWAMY, RAVI SANDHU, RAMOULI RAMASWAMY, AND RAVI S. Role-based access control features in commercial database management systems. In *In Proceedings of 21st NIST-NCSC National Information Systems Security Conference*, 1998.
- [90] V. RASTOGI, E. WELBOURNE, N. KHOUSSAINOVA, T. KRIPLEAN, M. BALAZINSKA, G. BORRIELLO, T. KOHNO, AND D. SUCIU. Expressing privacy policies using authorization views. In *Workshop on Ubicomp Privacy, (UbiComp)*, 2007.
- [91] J. REKIMOTO, Y. AYATSUKA, AND K. HAYASHI. Augment-able reality: situated communication through physical and digital spaces. *Wearable Computers, 1998. Digest of Papers. Second International Symposium on*, pages 68–75, Oct 1998.
- [92] MELANIE RIEBACK, BRUNO CRISPO, AND ANDREW TANENBAUM. The evolution of RFID security. *IEEE Pervasive Computing*, 2006.
- [93] THOMAS RISTENPART, GABRIEL MAGANIS, ARVIND KRISHNAMURTHY, AND TADAYOSHI KOHNO. Privacy-preserving location tracking of lost or stolen devices: cryptographic techniques and replacing trusted third parties with DHTs. In *Usenix Security Symposium*, 2008.
- [94] TOM RODDEN, ADRIAN FRIDAY, HENK MULLER, AND ALAN DIX. A lightweight approach to managing privacy in location-based services, equator-02-058. Technical Report CSTR-07-006, University of Nottingham and Lancaster University and University of Bristol, 2002.
- [95] JUNICHIRO SAITO AND KOUICHI SAKURAI. Grouping proof for RFID tags. In *AINA*, 2005.
- [96] CHIRAG SHAH AND W. BRUCE CROFT. Evaluating high accuracy retrieval techniques. In *SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 2–9, New York, NY, USA, 2004. ACM.
- [97] BO SHENG, CHIU C. TAN, QUN LI, AND WEIZHEN MAO. Finding popular categories for rfid tags. In *Mobihoc*, 2008.
- [98] DAVID SIMPLOT-RYL, IVAN STOJMENOVIC, ALEKSANDAR MICIC, AND AMIYA NAYAK. A hybrid randomized protocol for RFID tag identification. In *Sensor Review*, 2006.
- [99] DAWN XIAODONG SONG, DAVID WAGNER, AND ADRIAN PERRIG. Practical techniques for searches on encrypted data. In *IEEE Symposium on Security and Privacy*, 2000.
- [100] VINCE STANFORD. Pervasive computing goes the last hundred feet with rfid systems. *IEEE Pervasive Computing*, 2003.
- [101] T. STARNER, D. KIRSCH, AND S. ASSEFA. The locust swarm: an environmentally-powered, networkless location and messaging system. *Wearable Computers, 1997. Digest of Papers., First International Symposium on*, pages 169–170, Oct 1997.
- [102] CHIU C. TAN AND QUN LI. A robust and secure RFID-based pedigree system (short paper). In *ICICS*, 2006.
- [103] CHIU C. TAN, BO SHENG, AND QUN LI. Severless search and authentication protocols for rfid. In *International Conference on Pervasive Computing and Communications (PerCom)*, 2007.

- [104] CHIU C. TAN, BO SHENG, HAODONG WANG, AND QUN LI. MicroSearch: When search engines meet small devices. In *Pervasive*, pages 93–110, Sydney, Australia, May 2008.
- [105] CHIU C. TAN, HAODONG WANG, SHENG ZHONG, AND QUN LI. Body sensor network security: An identity-based cryptography approach. In *ACM Conference on Wireless Security (WiSec)*, 2008.
- [106] GENE TSUDIK. YA-TRAP: Yet another trivial RFID authentication protocol. In *PerCom*, 2006.
- [107] GENE TSUDIK. Ya-trap: Yet another trivial rfid authentication protocol. In *PERCOMW*, 2006.
- [108] ISTVÁN VAJDA AND LEVENTE BUTTYÁN. Lightweight authentication protocols for low-cost RFID tags. In *Ubicomp*, 2003.
- [109] HARALD VOGT. Efficient object identification with passive RFID tags. In *Pervasive*, 2002.
- [110] ELLEN M. VOORHEES. Overview of the trec 2001 question answering track. In *In Proceedings of the Tenth Text REtrieval Conference (TREC)*, pages 42–51, 2001.
- [111] HAODONG WANG AND QUN LI. Distributed user access control in sensor networks. In *DCOSS*, 2006.
- [112] HAODONG WANG AND QUN LI. Efficient implementation of public key cryptosystems on mote sensors (short paper). In *ICICS*, 2006.
- [113] HAODONG WANG, BO SHENG, AND QUN LI. Elliptic curve cryptography based access control in sensor networks. *International Journal of Sensor Networks*, 2006.
- [114] HAODONG WANG, BO SHENG, CHIU C. TAN, AND QUN LI. Comparing symmetric-key and public-key schemes in sensor networks. In *IEEE ICDCS*, 2008.
- [115] HAODONG WANG, CHIU C. TAN, AND QUN LI. Snoogle: A search engine for physical world. In *IEEE Infocom*, pages 1382–1390, Phoenix, AZ, April 2008.
- [116] SHUHONG WANG, XUHUA DING, ROBERT H. DENG, AND FENG BAO. Private information retrieval using trusted hardware. In *European Symposium On Research In Computer Security (ESORICS)*, 2006.
- [117] STEPHEN WEIS, SANJAY SARMA, RONALD RIVEST, AND DANIEL ENGELS. Security and privacy aspects of low-cost radio frequency identification systems. In *International Conference on Security in Pervasive Computing – SPC 2003*, 2003.
- [118] STEPHEN WEIS, SANJAY SARMA, RONALD RIVEST, AND DANIEL ENGELS. Security and Privacy Aspects of Low-Cost Radio Frequency Identification Systems. In *International Conference on Security in Pervasive Computing*, 2003.
- [119] DAVID WOODHOUSE. Jffs : The journalling flash file system. In *Proceedings Ottawa Linux Symposium*, 2001.
- [120] ZHIQIANG YANG, SHENG ZHONG, AND REBECCA N. WRIGHT. Privacy-preserving queries on encrypted data. In *European Symposium On Research In Computer Security (ESORICS)*, 2006.
- [121] KOK-KIONG YAP, VIKRAM SRINIVASAN, AND MEHUL MOTANI. Max: human-centric search of the physical world. In *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 166–179, New York, NY, USA, 2005. ACM.

- [122] DEMETRIOS ZEINALIPOUR-YAZTI, SONG LIN, VANA KALOGERAKI, DIMITRIOS GUNOPULOS, AND WALID A. NAJJAR. *Microhash: an efficient index structure for fash-based sensor devices*. In *FAST'05: Proceedings of the 4th conference on USENIX Conference on File and Storage Technologies*, pages 3–3, Berkeley, CA, USA, 2005. USENIX Association.
- [123] MUXIANG ZHANG AND YUGUANG FANG. Security analysis and enhancements of 3GPP authentication and key agreement protocol. *IEEE Transactions on Wireless Communications*, 2005.
- [124] YANCHAO ZHANG, WEI LIU, WENJING LOU, AND YUGUANG FANG. Location-based compromise-tolerant security mechanisms for wireless sensor networks. In *IEEE Journal on Selected Areas in Communications*, 2006.
- [125] LIN ZHONG, MIKE SINCLAIR, AND RAY BITTNER. A phone-centered body sensor network platform: cost, energy efficiency and user interface. In *BSN*, 2006.

## VITA

### Chiu C. Tan

Chiu C. Tan received his Bachelor of Science degree in Computer Science and Bachelor of Arts degree in Economics from the University of Texas at Austin in 2004. He was admitted to the Ph.D. program in Computer Science Department at the College of William and Mary in 2005, and became a Ph.D. candidate in 2006. His major research interests include security and privacy for cyber-physical systems, RFID systems, embedded systems, and wireless networks.