THESIS
2
2007

This is to certify that the
dissertation entitled

SECURE AND PRIVATE SERVICE DISCOVERY IN PERVASIVE COMPUTING ENVIRONMENTS

presented by

FENG ZHU

has been accepted towards fulfillment
of the requirements for the

Ph.D. degree in Computer Science and Engineering

[illegible]

Major Professor's Signature

June 15, 2006

Date

SECURE AND PRIVATE SERVICE DISCOVERY IN PERVASIVE COMPUTING ENVIRONMENTS

By

Feng Zhu

A Dissertation

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

DOCTOR OF PHILOSOPHY

Department of Computer Science and Engineering

2006

ABSTRACT

SECURE AND PRIVATE SERVICE DISCOVERY IN PERVASIVE COMPUTING ENVIRONMENTS

By

Feng Zhu

Service discovery is an essential element in pervasive computing environments to minimize administrative overhead. It enables devices and network services to properly discover, configure, and then communicate with each other. Much research on service discovery has been conducted, but most protocols are not designed with security and privacy support. Therefore, services may be discovered and used by anyone and subject to attacks. Directly applying the existing security and privacy solutions, however, may not work simply because the environments change.

This research addressed four areas of secure service discovery design, namely, unfamiliar environments, authentication, personal privacy, and its application for entity authentication in pervasive computing. The research resulted in a comprehensive survey, identification of essential secure and private service discovery issues, and novel design for convenient, secure, and private service discovery.

Our initial design of secure and private service discovery protocols targeted public environments. Proxy-based service discovery models were proposed to facilitate both mobile clients and services for secure and private service access and sharing. Then, we addressed a critical authentication issue in pervasive service discovery: involvement of necessary and proper users and service providers in a discovery session. We proposed a prudent service discovery model for users and service providers to establish mutual trust. The model also automates authentication processes. Therefore, users do not need to memorize the relationship among network services, service providers, and credentials. Next, we identified a difficult privacy challenge and expressed it as a "chicken-and-egg problem": both users and service providers want the other parties to expose sensitive information first. We proposed a progressive and probabilistic approach to protect privacy while achieving security and good usability at the same time. Our progressive exposure approach enables partial exposure. As a result, we provide an additional exposure choice besides the conventional two choices: expose or not expose. Last, we extended and applied prudent and progressive design to entity authentication for pervasive computing environments. A person uses a single device, the Master Key, for entity authentication. The Master Key aggregates one's digital access tokens and automatically discovers and selects proper tokens for authentication.

We built mathematical models, designed algorithms, provided theoretical analysis, and designed security protocols. We implemented protocols and built prototype systems. Security protocols were formally verified and analyzed. Threats and attacks were also analyzed, and counter attack measurements were proposed. In addition, much experimentation was conducted to verify our design, to test hypotheses, and to measure performance.

To my parents

ACKNOWLEDGMENTS

I am grateful to my advisor Dr. Matt W. Mutka. I have learned and continue to learn about doing research and writing from him. Without his guidance, I doubt that I would have published in major conferences and journals. His influence and encouragement made my Ph.D. study an extremely rewarding journey.

I am privileged as a graduate student at Michigan State University to learn computer science. Dr. Lionel M. Ni has been my advisor since I was a master student. His guidance and enthusiasm for research projects were a great help for me. Even after Dr. Ni moved to Hong Kong, he still offered much encouragement and advice.

I express my appreciation to Dr. Abdol H. Esfahanian, Dr. Li Xiao, and Dr. Zhengfang Zhou. They generously offered and shared their invaluable ideas, suggestions, experience, and comments with me.

I thank my parents, Xiling Zhu and Baimei Su. They helped me in the most difficult time during my graduate study. They supported and encouraged me throughout the journey of acquiring three advanced degrees. My mother inspired me to dedicate to my work and overcome obstacles. My father, a professor, gave me useful tips on research, writing, and career, which I tried to follow.

Finally, I thank my family for their support and love. My wife, Wei Zhu, was pursuing her Ph.D. degree during the same time. Whenever I wanted to work, she took care of everything else. She brainstormed about ideas and issues in my research projects and proofread my papers. My children, Anthony and Lillian, brought me joy. They were the most wonderful distraction. To all, my love and gratitude.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1. INTRODUCTION

In pervasive computing environments, a large number of networked computing devices, which range from tiny sensors to extremely dynamic and powerful devices, are gracefully integrated with people and their ambient environments. For example, a room may be saturated with hundreds of devices. The devices may provide people with information without needing their active attention.

To achieve such sophistication, service discovery is considered an essential step that enables devices and services to properly discover, configure, and then communicate with each other. Unfortunately, we now spend precious time actively finding services and manually configuring devices and programs. Sometimes special skills are necessary to perform the administrative overhead, which may have nothing to do with the tasks that we want to accomplish. System designers, on the other hand, may not foresee and code all possible interactions and states among devices and programs at the time of design. Service discovery protocols intend to reduce administrative overhead towards zero administration and therefore increase usability. Moreover, by adding a level of indirection, service discovery protocols simplify designs of pervasive systems.

Perhaps the most distinguishing characteristics of service discovery in pervasive computing environments are the integrations of computing devices with people and their ambient environments. Extremely dynamic environments and computing resources, mobility, and a wide range of heterogeneous devices, protocols, and platforms are also crucial issues and should be addressed [1].

Pervasive computing environments pose unique and critical challenges regarding security, privacy, and usability. People may own computing devices and share with others, and thus they become service providers. One may interact with hundreds or thousands of devices at different places and over the time. But many people do not have special skills to manage the devices.

1.1. Research Motivation

Secure and private service discovery are very important in pervasive computing environments. However, directly applying existing security solutions may fail simply because the environments change [2]. One may not only interact with a large number of network services, but many service providers as well. At a place, users and service providers might be even unfamiliar with each other. In addition, we want users to be able to access network service anywhere and at any time, yet we want them to access network services without spend their precious time on the system administrative overhead.

Moreover, when smart devices are embedded into our personal belongings and ambient environments, what we wear and carry are intelligent. Those devices provide useful information and may communicate over the wireless network. Therefore, personal privacy becomes a serious issue.

To help address the problems and solutions, we sketch a scenario of Bob, a physician, discovering services at different places. Bob's house has various wired and wireless computing devices. He shares these devices and services with his family members. As usual, he puts his cell phone, PDA, and MP3 player in his handbag and a Bluetooth earphone in his pocket, and then travels to his office. On the way to his office, he may wear his Bluetooth earphone and use it to discover his Bluetooth MP3 player and listen to songs. Nevertheless, he does not want others to know what is in his bag. In his office, he uses his computer, cell phone, and MP3 player. When he goes to Alice's office, they look at a document on the office file server simultaneously with their respective laptops via the office's wireless LAN. The devices within his pocket, however, should not be able to discover and use Alice's personal services on the devices in her purse, and vice versa, unless Alice later provides a user name and a password for him to access. Bob also volunteers to help patients at public places such as at a train station in case of emergencies. If a medical emergency happens and Bob is in the vicinity, he is notified of the patient's position. Bob may follow the directions on a map shown on his PDA and moves towards the patient. At the place, Bob reads the medical information from a medical device that the patient wears. When Bob is in an airport and he has an hour before his flight leaves. He turns on his PDA and finds that there is a wireless LAN available. After Bob makes the connection, he receives an email that includes an attached document. Then, he uses his PDA to search for a nearby printer to print the document so he can read it during his flight.

We identified that the following information is sensitive should be protected during service discovery:

- At the service owners' side, service information, owners' identities, and presence information.
- At the users' side, identities used for authentication, user's presence information, and service query information.

Achieving the two goals is challenging in pervasive computing environments. First, services coexisting in a place may belong to different owners. For example in Bob's work place, services belong to Bob, Alice, or the office. Second, user mobility and service mobility cause the available domains and services to change dramatically. For instance, different services are available in Bob's house and on his way to the office. Bob carries his mobile devices, and the services on them move with Bob. Third, since users act in many different roles, they use different identities for different administrative domains. For instance, Bob uses one user name to access his office PC and another user name for his PDA. Fourth, sensitive information should be discovered, accessed, and exposed only when it is necessary, for instance, a medicate device that one wears.

In summary, we want to provide convenient, secure, and private service discovery protocols for users to discover and use network services. Our design goals are: users discover and access network services without administrative overhead; only legitimate users can discover and access network services and devices; and personal privacy is protected and only exposed when it is needed.

1.2. Contributions

This dissertation contributes to the research of service discovery for pervasive computing and security for pervasive computing. Particularly, the research contributes in the following five aspects:

- *Comprehensive surveys.* The research resulted one of the most comprehensive survey of service discovery and secure service discovery for pervasive computing environments so far. The research also produced a survey of entity authentication using digital credentials.
- *Service discovery in unfamiliar environments.* Unlike the existing approaches, we propose a new service discovery model, called Splendor, supporting nomadic

users and services in public environments. Splendor emphasizes secure and private service discovery in such environments. Location awareness is integrated for location dependent services discovery and is used to lessen service discovery network infrastructure requirements. In addition, we proposed a proxy-based approach for ad hoc wireless networks. By using hidden network channels, unfamiliar parties establish trust relationships and securely discover and user network services.

- *Authentication for pervasive service discovery.* We present a user-centric model, called *PrudentExposure*, which exposes minimal information privately and securely. Users and service owners exchange code words in an efficient and scalable form to establish mutual trust. Based on the trust, secure service discovery sessions are set up. The model automates authentication processes. Therefore, users do not need to remember the relationship among network services, service providers, and credentials. Such relationship may be very complex and dynamic in pervasive computing environments.
- *Personal privacy protection.* We identify that privacy protection may be as difficult as a chicken-and-egg problem, in which both users and service providers want the other parties to expose sensitive information first. We propose a progressive and probabilistic approach to solve the problem. Users and service providers expose partial information in turn and eliminate unnecessary exposure during the process. The approach can be used beyond secure and private service discovery. Our progressive exposure approach provides an addition choice besides the conventional two exposure choices: expose or not expose.

- *Entity authentication in pervasive computing.* We propose a novel entity authentication approach for pervasive computing environments. A person uses a single device, the Master Key, which aggregates all his digital forms of access tokens for entity authentication. The Master Key discovers and selects proper tokens for its owner. With an emphasis on usability, the Master Key secures authentication, protects privacy information from outsiders and insiders, and supports various claimant-verifier relations.

1.3. Dissertation Outline

In Chapter 2, we discuss relate service discovery research and authentication in pervasive computing environments. We present a classification of existing service discovery protocols including secure service discovery design. We also discuss authentication in pervasive computing and entity authentication via digital tokens.

Chapter 3 illustrates the design of Splendor service discovery. Splendor focuses on mobile and unfamiliar parties to authenticate each other and establish service usage. In Chapter 4, we present a proxy-based approach that uses alternative network channels to facilitate ad hoc wireless communications and to establish a secure trust relationship between unfamiliar parties.

Next, in Chapter 5, we show our prudent service discovery design in pervasive computing environments. A secure and private authentication for pervasive service discovery together with a new service discovery model is the base for the design, improvement, and extension in the later chapters. Chapter 6 demonstrates a progressive exposure approach to improve and protect personal privacy. In Chapter 7, we further extend and apply the prudent and progressive design to entity authentication in pervasive computing environments. Last, in Chapter 8, we conclude this dissertation research and point out our future work.

CHAPTER 2. BACKGROUND AND RELATED WORK

In this chapter, we first provide a survey and a taxonomy of major service discovery protocols. Then, we discuss secure and private service discovery design for pervasive computing environments and some representative protocols. Last, we discuss authentication and entity authentication for pervasive computing environments.

2.1. Service Discovery in Pervasive Computing Environments

Service discovery protocols are designed to minimize administrative overhead and increase usability. They can also save pervasive system designers from trying to foresee and code all possible interactions and states among devices and programs at design time. By adding a layer of indirection, service discovery protocols simplify pervasive system design.

Over the last few years, many service discovery protocols were designed and developed. Examples in academia include Intentional Naming System (INS) [3] at MIT, Ninja Service Discovery Service (SDS) [4] at UC Berkeley, DEAPspace [5] at IBM research. Major software vendors ship their service discovery protocols with their current operating systems, for example, Jini Network Technology [6] from Sun Microsystems, Universal Plug and Play (UPnP) from Microsoft [7], and Rendezvous [8] from Apple Computer. Different organizations also proposed service discovery protocols and are trying to make them standards. The Salutation protocol is developed by Salutation Consortium [9]; Service Location Protocol (SLP) [10] is a standard track protocol posted by IETF; and Bluetooth Service Discovery Protocol (Bluetooth SDP) [11] from the Bluetooth Special Interest Group is part of the Bluetooth specification.

All protocols support discovery of services in ambient environments (in terms of network topology or location), but each of these service discovery protocols addresses a different mix of issues in service discovery. Most protocols are designed for home or enterprise environments, and therefore applying them directly to pervasive computing environments may not be appropriate. Various design approaches, however, may be used as valuable references for the future design of service discovery protocols for pervasive computing environments.

2.1.1. Pervasive Computing Environment Challenges

Pervasive computing environments are far more dynamic and heterogeneous than enterprise environments. Enterprise network services operate within a network scope protected by firewalls and managed by system administrators. But a network scope can't easily define the ambient services in pervasive computing environments, nor are all services likely to be managed by system administrators. Pervasive service discovery in an office environment might target an enterprise computing service, but it might also target a service that doesn't belong to the enterprise domain—for instance, a colleague's personal service.

Discovery of web services over the Internet seems opposite to discovery of ambient services. The former has no physical location limitation, while the latter only cares about services in the vicinity. Moreover, current designs of the web service architectures are only focused on interoperations among applications; while for pervasive computing, integration of computing devices with people is critical. A web service architecture provides a common definition to facilitate the interoperability among services running on various platforms [12], namely based on the Web Service Description Language

(WSDL), SOAP, Hyper Text Transform Protocol (HTTP), and eXtensible Markup Language (XML). Universal Description Discovery and Integration (UDDI) [13] is a representative web service discovery protocol. Service discovery via UDDI may involve analysts, programmers, and administrators. Moreover, with its focus on electronic commerce, UDDI's registry and data structure designs are too specific for services in pervasive computing.

Pervasive computing environments pose unique questions relative to integration with people and their ambient environments.

2.1.2. Integration with People

Perhaps the most serious challenge to pervasive computing service discovery is the integration of computing devices with people. For one thing, how do we protect personal privacy? When computing devices are integrated with people, much personal information is expressed in digital form. Service discovery protocols can communicate this information over networks—either directly or by inference. For example, you can infer a person's presence information from the RFID tags on clothes or the devices carried in a hand bag. Similarly, you can infer a person's health status from discovering a wearable medical device. Moreover, users expose their intentions in service discovery—to untargeted as well as targeted services. These possibilities impose complex requirements on system design.

Another question has to do with how much prior knowledge a user or service provider must have for service discovery. Most existing protocols consider only interactions among computers and programs that are acting as clients (discovering devices), services (network services), and directories (centralized service information storage).

Nevertheless, people serve two roles in discovery processes: as users of services via clients and as service providers. One end of the spectrum requires little knowledge — users can acquire information without actively managing pervasive computing devices. The other end requires special skills. For instance, many existing service discovery protocols standardize service names and attributes to eliminate ambiguity when, say, a client looks for a "print" service while a service provides a "printing" service. Does this mean that a user must learn service terminology?

Finally, pervasive computing environments allow multiple service providers to coexist at a place. What user roles and administrative domains are properly involved in a discovery? For example, in Bob's office, he might own services and so might his colleague Alice, the office, or the landlord. The credentials he uses to access Alice's MP3 player differ from those he uses to access the office printer. You can't assume that users will provide proper credentials as they do in traditional computing environments because supplying them requires knowledge of the relations among services, service providers, and the user's credentials. The relations can be complex or ultimately unmanageable when a user interacts with thousands of dynamic services that are owned by dozens of service providers.

2.1.3. Integration with Environments

Integrating computing devices with environments raises an interesting question: how can we precisely define the ambient environment that a service discovery instance targets?

The primary service discovery protocol approaches are based on LANs and single-hop wireless communication[6, 7, 11], These approaches are simple and effective for, respectively, enterprise environments that are behind firewalls and devices that communicate via ad hoc networks. However, they can be too coarse for pervasive computing environments, which are more appropriately defined by physical-world boundaries [1], such as a room, and other high-level context information, such as location and current user tasks. Moreover, different users in the same place can have different views of the augmented environments. For instance, a visitor's view differs from a host's view.

In pervasive computing, heterogeneity occurs in many aspects: hardware, software platforms, network protocols, and service providers. Future devices and services will increase the diversity. Although service discovery protocols accommodate heterogeneity, existing protocols have various design goals and solutions. Each has its advantages and disadvantages in different situations, so it seems unlikely that a single protocol could dominate in pervasive computing environments. With current protocols, this means clients and services can't discover each other if they don't use a common protocol. We should therefore establish a common platform to enable interoperability among service discovery protocols.

Are the dynamic conditions typical in pervasive computing qualitatively different from those in mobile computing systems? They seem at least to reach an extreme in pervasive computing: unattended tiny devices, user and device mobility, changing user roles and tasks, and partial failure on devices and applications. Service discovery results can be inconsistent in consecutive queries, and service states can be uncertain. When inconsistency and uncertainty rates are high, users may perceive the system as unstable. Many service discovery protocols adopt time-based approaches (soft states and lease-based approaches) to address dynamic conditions. These approaches simplify distributed system designs to handle dynamic conditions. However, quantitative design analysis might be difficult without actual pervasive computing environments to measure.

2.2. Service Discovery Protocols

2.2.1. Bluetooth Service Discovery Protocol

Bluetooth, from the Bluetooth Special Interest Group (SIG), enables nearby devices to communicate with each other at low cost and low power consumption [14]. Part of the Bluetooth specification is the Bluetooth Service Discovery Protocol, which enables Bluetooth devices to discover each other.

2.2.2. DEAPspace

Researchers at IBM Research have studied and proposed a service discovery protocol for single-hop ad hoc environments, known as DEAPspace [5]. In contrast to other service discovery protocols in which services announce their information, DEAPspace's algorithm caches service information at each node, then each node broadcasts its knowledge of other services and its own services in turn (we call this cache-broadcast approach). The nodes learn from others. Service lookup is accomplished by searching the local cache. Furthermore, energy weak devices use idle mode to save power.

2.2.3. Intentional Naming System (INS) and INS/Twine

Researchers at MIT designed INS [3]. As services in INS are not mapped to fixed service locations, a level of indirection is added. INS resolves a service lookup to a service location at the delivery time, known as *late binding*.

In INS/Twine [15], service attributes are hashed and stored in mesh structure directories. Service lookup is based on peer-to-peer technology, a more scalable approach to handle millions of services. Service lookups, however, may go through several directories, which may have additional search latency.

2.2.4. Jini Network Technology

Sun Microsystems' Jini is based on Java technology [16]. One special feature of Jini is the mobile Java codes, which may be moved among clients, services, and directories. The advantage of Jini is its platform independency, but the disadvantage is that all the clients, services, and directories depend on Java runtime environments directly or indirectly.

2.2.5. Salutation

The Salutation Consortium has rolled out the Salutation protocol [9]. It is an open source protocol and royalty-free. One advantage of the protocol is that it implements two interfaces. One interface is for applications. The other interface is designed to be independent of the transport layer, so that it is very flexible to use various underlying transport protocols and may be used for more environments. Furthermore, a mapping of Salutation over the Bluetooth Service Discovery Protocol has been specified [17].

2.2.6. Secure Service Discovery Service (SSDS)

SSDS at UC Berkeley puts emphasis on security and supporting a huge number of services, known as wide-area support [18]. Authentication between clients and services is based on certificates. Public key and symmetric key encryption are used for confidentiality and communication data privacy. Message Authentication Code (MAC) is used to ensure message integrity. Services manage their access control lists for their users and publish on servers, known as capacity servers. For wide-area support, different hierarchical directory structures are considered. A technology based on the Bloom filter [19] is used to achieve service information aggregation and filtering when building up the hierarchical directories.

2.2.7. Service Location Protocol (SLP) Version 2

Posted by IETF as a standard track protocol, SLP is for enterprise environments [10]. As the protocol name states, SLP only defines a way to locate a service and leaves open the interaction between clients and services after service discovery. URLs are used for service locations.

2.2.8. Universal Plug and Play (UPnP)

UPnP is from the UPnP Forum [20]. The major player is the Microsoft Corporation. UPnP targets unmanaged networking environments, such as home environments. UPnP is a device oriented service discovery protocol. All the service information and communication is in the XML format, which is platform and programming language independent and greatly increases interoperability between devices.

2.2.9. Rendezvous

Apple Computer's Rendezvous is an DNS-based service discovery protocol [21]. It uses the existing DNS resource record types to name and assist service discovery. Rendezvous is also known as Zero Configuration networking (Zeroconf) [22]. The advantage of Rendezvous is the ubiquity of DNS servers.

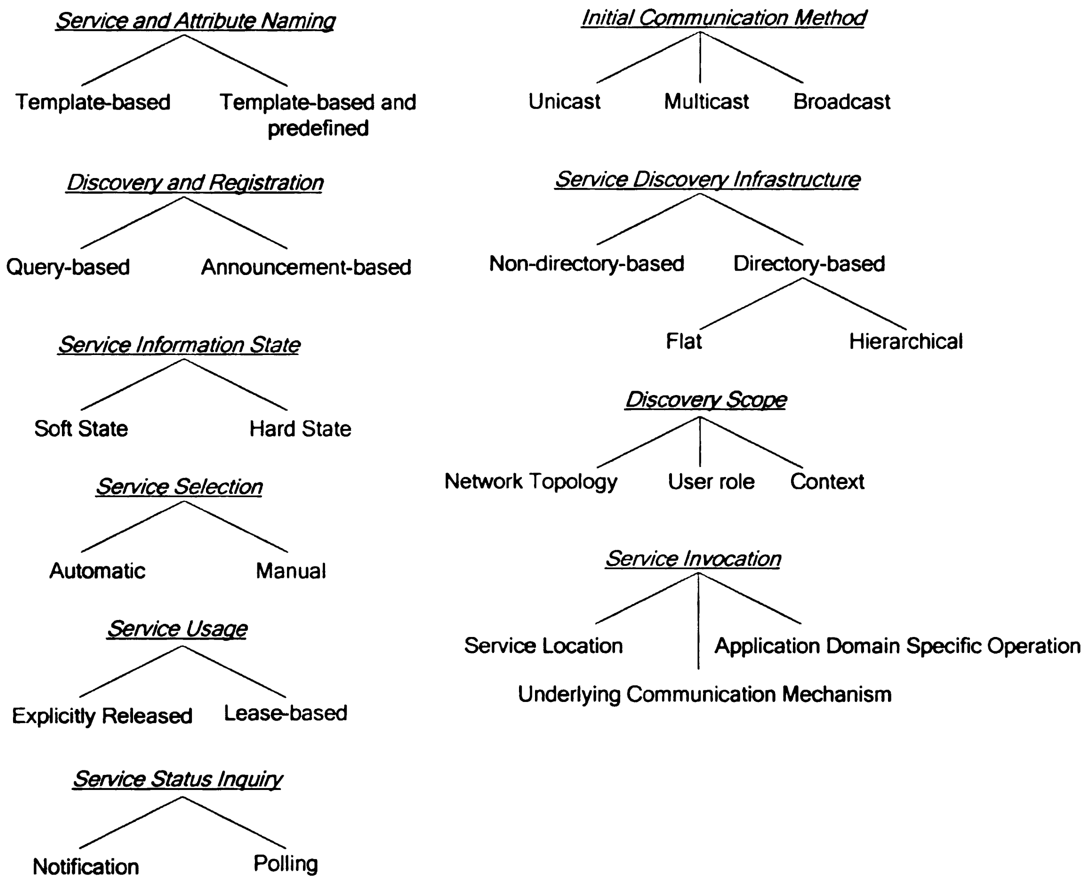

Figure 2-1. Major components of service discovery protocols and their classifications.

2.3. Service Discovery Designs

Existing service discovery protocols have their respective target environments and design goals, and thus they emphasize different aspects of service discovery. However, many service discovery protocols are for LAN or ad hoc network environments in which service discovery in pervasive computing environments is conducted. A comparison of the designs of service discovery protocols facilitates future designs for pervasive computing. Although protocols may use different terminologies to better reflect their design choices, a common terminology and classification mechanism assist the analysis of their advantages and disadvantages and identify areas worthy of additional efforts.

Figure 2-1 shows major components of service discovery protocols and a classification of the different design choices of the components. In Table 2-1, Table 2-2, and Table 2-3, we present a comparison of service discovery protocols. Our discussion of the components follows the procedure of a service discovery process.

2.3.1. Service and Attribute Naming

In a service discovery process, a client searches a service by specifying the name and/or attributes. This approach has much lower administrative overhead than the traditional approach that requires a priori knowledge of the existence of a service and manual inputs such as computer names. In pervasive computing environments, there may be hundreds of devices and services in a place. It is overwhelming to manually configure all devices and services for potential service accesses.

To provide expressiveness for various services and expandability for new services, many service discovery protocols use template-based naming; that is a protocol defines the format of service names and/or attributes. For instance, Rendezvous defines how a service name is composed. Besides template-based naming, several protocols also predefine a set of common attributes and/or frequently used service names. For example, Bluetooth SDP defines a set of 128-bit Universal Unique Identifiers for frequently used services and a set of attribute IDs. When clients, services, and directories interact with each other, there is no ambiguity regarding the predefined names and attributes.

Several protocols, such as Jini and Bluetooth, offer user-friendly service names and attributes besides the machine-friendly versions. Nevertheless, users still need to learn the terms in order to use services.

Table 2-1. Comparison of service discovery protocols.

	Bluetooth SDP	**DEAPspace**	**INS**
Service and attribute naming	Template-based and predefined	N/A	N/A
Initial communication method	Unicast and broadcast	Broadcast	Unicast, and multicast
Discovery and registration	Query	Announcement	Query and announcement
Service discovery infrastructure	Non-directory-based	Non-directory-based	Directory-based Flat or hierarchical
Service information state	Soft state	Soft state	Soft state
Discovery scope	Network topology (Single hop ad-hoc network)	Network topology (Single hop ad-hoc network)	User role (Administrative domain)
Service selection	Manual	Manual	Automatic
Service invocation	Service location	N/A	N/A
Service usage	N/A	N/A	N/A
Service status inquiry	N/A	N/A	N/A
Security dependency	Built-in and Bluetooth Security	N/A	N/A
Authentication	Yes (Device authentication)	N/A	N/A
Authorization	Yes	N/A	N/A
Confidentiality	Yes	N/A	N/A
Integrity	No	N/A	N/A
Availability	No	N/A	N/A
Non-repudiation	No	N/A	N/A
Privacy	No	N/A	N/A

Table 2-2. Comparison of service discovery protocols.

	Jini	Ninja SDS	Rendezvous
Service and attribute naming	Template-based and predefined	N/A	Template-based
Initial communication method	Unicast and multicast	Unicast, multicast, and broadcast	Unicast and multicast
Discovery and registration	Query and announcement	Query and announcement	Query
Service discovery infrastructure	Directory-based Flat or hierarchical	Directory-based Hierarchical	Directory-based Hierarchical
Service information state	Soft state	Soft state and hard state	Soft state and hard state
Discovery scope	User role (Administrative domain), context (location), and network topology (LAN)	User role (Administrative domain), context (location), and network topology (LAN)	User role (Administrative domain)
Service selection	Manual	Manual	Manual
Service invocation	Service location and communication mechanism (Downloadable Java code and RMI)	N/A	Service location (TCP/IP address and port)
Service usage	Lease-based	N/A	N/A
Service status inquiry	Notification and event agent	N/A	N/A
Security dependency	Built-in	Built-in	DNS Security Extension
Authentication	Yes (User and service)	Yes (User and service)	Yes (Service information)
Authorization	Yes	Yes (Access control list and capability)	No
Confidentiality	No	Yes	No
Integrity	Yes	Yes	Yes
Availability	No	Yes	No
Non-repudiation	No	Yes	No
Privacy	No	Service privacy	No

Table 2-3. Comparison of service discovery protocols.

	Salutation	**SLP**	**UPnP**
Service and attribute naming	Template-based and predefined	Template-based	Template-based and predefined
Initial communication method	Unicast and broadcast	Unicast and multicast	Unicast and multicast
Discovery and registration	Query and announcement	Query and announcement	Query and announcement
Service discovery infrastructure	Directory-based Flat	Directory-based	Non-directory-based
Service information state	Hard state with periodically check	Soft state	Soft state
Discovery scope	Network topology (LAN)	User role and network topology (LAN)	Network topology (LAN)
Service selection	Manual	Manual	Manual
Service invocation	Service location, communication mechanism (RPC), and operations	Service location (URL)	Service location, communication mechanism (URL, SOAP, and HTTP), and operations
Service usage	Explicitly released	Explicitly released	Explicitly released
Service status inquiry	Notification	N/A	Polling and notification
Security dependency	Built-in	Built-in	UPnP Security
Authentication	Yes (User)	Yes (Service information)	Yes (User, service, and device)
Authorization	Yes	No	Yes (Access control list and certificates)
Confidentiality	No	No	Yes
Integrity	No	Yes	Yes
Availability	No	No	No
Non-repudiation	No	No	No
Privacy	No	No	No

Given the various templates and predefined service names and attributes, a problem for the pervasive computing environment is: if more than one service discovery protocol is widely adopted, clients, services, and directories have to understand every protocol in order to discover each other.

2.3.2. Initial Communication Method

The most efficient initial communication method among clients, services, and/or directories is unicast because only related parties are involved. The drawback of unicast is that network addresses need to be configured with a priori knowledge. Initial configuration of many devices for potential communications may be tedious or even infeasible in pervasive computing.

An elegant solution in current service discovery protocols is UDP multicast. After a few multicast messages, unicast addresses are dynamically determined and communications are switched to unicast. In this way, only a few multicast addresses need to be established initially on clients, services, and directories. The unicast addresses may be saved for future usage and performance optimization.

An alternative method is link layer broadcast, as done in Bluetooth SDP and Salutation. Broadcast has the same advantage as multicast, but it is usually limited within a single hop of wired or wireless networks.

When a service discovery protocol is tightly bound to the underlying network protocols, its discovery capability may be limited. For example, a device with only a Bluetooth network interface will not be discovered by any other devices without a Bluetooth network interface. An innovative solution in Salutation provides two interfaces: one for applications to call service discovery functions and the other to make the Salutation transport layer independent. A mapping from Salutation to Bluetooth SDP enables discovery of Bluetooth devices [17].

2.3.3. Discovery and Registration

There are two basic approaches for clients, services, and directories to exchange discovery and registration information: announcement-based and query-based approaches. In the announcement-based approach, interested parties listen on a channel. When hearing an announcement, all parties learn the information. For example, when a service announces its information, a client may learn its existence and a directory may register the service information. In the query-based approach, a party will receive an immediate response and does not need to process unrelated announcements, but multiple queries may ask for the same information and will be answered separately. Many protocols support both approaches.

2.3.4. Service Discovery Infrastructure

At the first level in the hierarchy, a choice is made between the directory-based or non-directory-based model. The directory-based model has a dedicated component – a directory that maintains service information and processes queries and announcements. Some directories provide additional functionality, for instance the directories in Ninja SDS support secure announcements and queries. The non-directory-based model does not have a dedicated component. When hearing a query, every service processes the query and replies back if it matches the query. When hearing a service announcement, a client may record service information for future use. The non-directory-based model is suitable for simple environments such as home environments where there are a few services. The directory-based model is suitable for environments where there are hundreds or thousands of services because a directory processes clients' queries more efficiently than if all services are involved.

Jini, SLP, and Salutation provide more flexible solutions. Jini and SLP support both the directory-based and non-directory-based models. Although there is some overhead to determine the existing discovery model in the initial discovery stage, such flexibility is preferable in pervasive computing environments because of heterogeneity. Salutation has a component called Salutation Manager, which integrates the service discovery functions of the client, service, and directory into one component. Therefore, one device may act as different roles in different situations.

At the next level in the hierarchy (beneath the directory-based model), directory designs fall into two categories: a flat structure and a hierarchical structure. In a flat directory structure, directories have peer-to-peer relationships. For example, INS has a flat mesh structure within its domain; and a directory exchanges information with all other directories. While in a hierarchical directory structure, directories form tree-like hierarchical structures, for instance, Rendezvous (based on the Domain Name System) and Ninja SDS. In pervasive computing environments, services may be integrated with ambient environments or people: a directory needs to determine whether the service information that it exchanges with a neighbor or a parent directory is appropriate.

2.3.5. Service Information State

Most service discovery protocols maintain service status as soft states. In a service announcement, the life span of the service is specified. Before the service expires, it should announce itself again to renew the lease on the registration, or a client or directory polls the service for validity. Otherwise, the service will not be valid to use after expiration. In the directory-based model, expired service entries will be removed from the directories. This graceful soft state service management mechanism greatly simplifies the system design and makes service information fresh. Conversely, service status can be hard state. By using hard state, few service announcements and housekeeping jobs are required. However, there is no guarantee that any service information is up-to-date unless the services are polled periodically.

2.3.6. Discovery Scope

Proper discovery scopes keep unnecessary computation minimal on clients, services, and directories. Network topology based, user role based, context information based, or a combination of the information help to properly define the target scope for a service discovery session.

In network topology based discovery, some protocols use LANs as their default discovery scope, while others use a single hop wireless network range as their scope. An implicit assumption is that the clients, services, and/or directories are within a place that belongs to the same administrative domain. The assumption may be true within enterprise computing environments and home environments, but it may not hold within pervasive computing environments. On the contrary, multiple administrative domains may coexist and have different underlying networks that may not be connected.

Using user roles as a discovery scope is another approach. Currently, many service discovery protocols support administrative domains as a discovery scope. Either users authenticate with a domain or supply the designated domain as an attribute. The advantage is that users can control the target domain, but the disadvantage is that users need a priori knowledge of the target service and its domain. A fine-grained approach should reflect an ambient environment based on a user's role instead of all users of a domain discovering the same augmented environment.

High level context information such as temporal, spatial, and user's activity could be very useful information to define the discovery scope. Proper usage of context information will save user's precious time and effort in discovering services. However, context information is integrated to few protocols and in very simple forms. For example, in Jini service discovery, physical location may be specified as an attribute of a service. In brief, graceful integration with context-aware systems and utilizing imperfect context information still need much research efforts.

2.3.7. Service Selection

Although discovery scopes limit the number of matched services, a discovery result may still contain a list of services. Manual and automatic service selections are the two extremes that a service discovery protocol may provide. Protocols that support manual service selection prompt users with lists of services and then let them select. Users control the target services, but the selection may be tedious and users may not have enough a priori knowledge to distinguish among the services.

To the other extreme, service discovery protocols may select a service for a user. The advantage of protocol selection is that it simplifies client programs or little user involvement is needed. In INS, a service lookup is resolved to a service location at the delivery time. When a set of similar services meet a client's request, a service request will be routed to one of the services based on the application-defined service weight. This is called *anycast* in INS. This approach utilizes the service side knowledge, and thus service load may be well balanced among services. Nevertheless, the target service may not meet the user's intent because it is challenging to find a service automatically and accurately that best fits the user's need when a user does not explicitly specify all requirements as attributes. A sophisticated service selection mechanism, Matchmaking with ClassAds in the Condor project [14], supports attributes matching even if some attributes are not explicitly given. Moreover, it allows both discovering parties and service providers to specify their preferences. In short, it can express a wide range of resource information without a schema.

2.3.8. Service Invocation

After a service is selected, a client invokes a service. Invocation involves a service's network address, an underlying communication mechanism, and application domain specific operations. Existing service discovery protocols provide three different levels of support. At the first level, only services' network addresses are provided, and applications are responsible to define the communication mechanism and operations. At the second level, in addition to service network addresses, a service discovery protocol defines the underlying communication mechanisms. Remote Procedure Call (RPC) and its variations are used. Appealing approaches include Jini and UPnP. Jini uses Remote Method Invocation and downloadable Java code. The Java code moves from a service to a directory and then to a client. Next, the client calls the service via the downloaded code. UPnP achieves service invocation based on the XML, SOAP, and HTTP. The platform-independent protocol stack in UPnP increases interoperability. WSDL is based on the similar underlying communication mechanism: XML, SOAP, and HTTP [23]. Besides, it provides a model to describe operations, message flows, fault handling, and plug-in components. At the third level, application domain specific operations are defined in addition to the communication mechanism and network address, such as done in Salutation and UPnP.

In pervasive computing, a client and a service may not have a priori application domain knowledge to understand and interact with each other, and thus a more general approach may be necessary. This is very useful because a general interface provides compatibility when new devices and services are introduced in the future. Pioneer work such as HP Cooltown [24] uses web pages as the general information format, and clients and services exchange URLs for addressing. Therefore, it requires minimal a priori knowledge. Researchers at PARC proposed a concept called recombinant computing [25]. It provides a set of general interfaces for arbitrary devices or services to interact with each other. In their implementation, the interfaces allow services and devices to setup connections and control behaviors. Devices and services understand each other via explicitly exchanging their context information. The advantage of this approach is that users may decide what devices or services should interact instead of application designers making the decisions.

2.3.9. Service Usage

In most service discovery protocols, a client needs to explicitly release resources of a service once a service access is granted. Unlike other protocols, Jini uses lease-based service usage. A client and a service negotiate a service usage period, which can be cancelled or renewed later. Thus, clients and services know resources will be reclaimed when leases expire. Since information will be deleted after leases expire, services will not accumulate stale information as may occur in traditional distributed systems when a client crashes. Therefore, in pervasive computing environments, the lease-based service usage is beneficial.

2.3.10. Service Status Inquiry

During a service invocation, a client may be interested in a service's events or status. One way of knowing about them is by polling the service periodically. Another way, known as service event notification, is that clients register with a service and then the service notifies clients who have shown interest. Most existing protocols implement service event notifications. However, when events are generated very frequently or a service status changes quickly, it is better to use service polling.

It is even better to have agents do event filtering and aggregation. Jini provides several such methods. Services send events to agents and let agents make sure all the events are delivered to clients; an agent may act as a sink for events, which will be filtered, aggregated, and then sent to clients; or an agent may resemble a mailbox to filter events over time. Although clients and services benefit from event filtering and aggregation, some computers in networks need to provide resources to handle the events.

2.3.11. Heterogeneous Device Support

Many devices in pervasive computing environments have limited resources, and thus do not have service discovery capabilities. Jini Surrogate [26] is designed to assist those devices to join Jini service discovery. A Jini capable machine hosts a Java object (surrogate) to represent a device for Jini service discovery.

2.4. Secure and Private Service Discovery Design

Without security, anyone may access anyone else's devices and services via service discovery protocols. Nevertheless, directly using existing security solutions with service discovery protocols may not be appropriate. Anderson pointed out in [2] that many security solutions failed when applying them to different environments, simply because the environments change. Furthermore, most people without special skills may need to manage and secure their personal devices and services as service providers. Unlike an enterprise environment where outsiders and insiders can be separated via physical boundaries and firewalls, in a pervasive computing environment, users and services providers may coexist and interact with each other via wireless networks. In addition, personal privacy is critical. From a service provider's perspective, service information, identities, and presence information may be sensitive. From a user's perspective, credentials used for authentication, presence information, and service query information may be sensitive.

While keeping in mind that environments change, we revisit the service discovery components. In discovery and registration, sensitive information should only be exchanged with *legitimate* parties in the vicinity for security and privacy. Legitimacy means not only valid credentials to verify identities but also in terms of services. For example, Bob's office may be a legitimate service provider to him in terms of printing services, but it may not be a legitimate service provider in terms of pop songs. Unfortunately, legitimacy may not be easy to acquire. Users that are discovering services may not have knowledge of existing services and service providers before discovery. Similarly, services and/or directories may not have knowledge of existing legitimate clients, services, and directories in the vicinity because of the mobility and dynamic characteristics of the latter. In [20], the legitimacy problem is expressed as a chicken-and-egg problem and a progressive exposure approach was proposed.

When rethinking about service discovery infrastructures from users' and service providers' perspectives, much improvement is necessary for security and proper interaction between human and computing devices. Current designs focus on interactions among clients, services, and directories. From a user's perspective, however, a discovery should reflect a user' role instead of a client device. For example, Bob and his son should discover different services at home when using the same PDA. More challengingly, a user may use a client device to interact with many services, which may require different kinds of credentials. Moreover, every person may own many services and become a service provider when computing devices are integrated with her belongings. From service providers' perspective, managing and sharing services via a service discovery protocol require system software to provide simple user interfaces and powerful tools to manage security and privacy. Ellison drew a conclusion that a security policy for a home environment can be much more complex than for an enterprise environment [27]. A more challenging and desired tool will allow users to learn who has accessed or is accessing a service.

Since service invocation may happen via insecure networks, a secure communication mechanism is needed to protect interaction between clients and services. Additionally, prudent design and implementation of the interfaces on clients and services are necessary because it is difficult to predict all potential counterparts that a client or a service may interact.

In comparison to service discovery functionality, security and privacy support in existing service discovery protocols are still at an early stage. In Table 2-1, Table 2-2, and Table 2-3, we compare service discovery protocols on authentication, authorization, confidentiality, integrity, availability, non-repudiation, and privacy. Most secure protocols shown in the table directly use existing security solutions, and thus many requirements that we discussed in this section are not met. However, recent revisions of the service discovery protocols and new protocols support more security features. The next section describes representative secure service discovery protocols.

2.5. Secure Service Discovery Protocols

Bluetooth Security [28] defines a profile for service discovery applications besides the security that Bluetooth provides at the link layer. Service discovery is classified as trusted and untrusted. In trusted discovery, service information is only exposed to a device that shares a common secret with the service. In untrusted discovery, service information is available to any device. In pervasive computing, requiring two parties to share a secret beforehand may be inconvenient or even infeasible. Thus, a high level protocol to distribute secrets to Bluetooth devices may be desired.

Unlike other secure service discovery protocols, UPnP Security defines interactions not only among computers but also among computers, people, and environments [29]. For example, it specifies a security procedure for a user to take ownership of a device. An important impact is that standardizing the human computer interaction procedure reduces many security problems caused by human operations. Moreover, since UPnP Security is designed for heterogeneous devices and services, it supports many authorization methods: access control lists, authorization servers, authorization certificates, and group definition certificates. In short, UPnP Security addresses many security issues for service discovery in pervasive computing.

Jini introduces a new service invocation approach, namely via downloadable Java code. The downloadable code raises new security issues: verifying trustworthiness of the code and granting permissions to it. In addition to addressing the new issues, Jini provides an abstract and extensible interface via which various security protocols may be used to support other security requirements such as authentication and confidentiality [30].

Ninja SDS is a centralized security solution. It supports authentication, authorization, data and service privacy, and integrity. Directories, known as Service Discovery Service servers, are trusted. Clients and services authenticate with the directories for service lookups and announcements, respectively. Ninja SDS is good for enterprise environments where users and services are willing to expose information to central directories. In pervasive computing, however, trusted central directories have the personal privacy problem because every service registers with central directories and every request is sent to central directories.

PrudentExposure addresses security and privacy issues when multiple service providers coexist [31]. The discovery is user-centric, that is the discovery is based on user roles. By exchanging little information, an agent selects relative credentials to authenticate with service providers automatically. The design simplifies the security requirements on clients and frees a user from memorizing the relationship among services, service providers, and his credentials. Much sensitive information is protected by not responding to arbitrary requests unless some trust information is exchanged.

2.6. Authentication in Pervasive Computing

The Resurrecting Duckling security policy [32] provided a new way for authentication in pervasive computing environments. By mimicking the behavior of mother ducks and ducklings, the policy set up a master-slave relation. The master-slave relation limited devices to talk to peers. Therefore, Stajano proposed additional research [33] to enable peer communications. The basic idea was that master devices might define policies, which allowed other devices to set up temporary master-slave relations to control the slave devices. The authors were the first who proposed the idea to use physical contact to exchange a secret before two devices set up secure wireless communications.

Balfanz, et al. at Xerox Palo Alto Research Center, extended the Resurrecting Duckling security policy [34]. Their work tried to solve the authentication problem for securely using a public printer at an airport without using a public key infrastructure and universal naming convention for printers. To locate a printer, they suggested walking to the printer instead of typing a URL in a web browser as an alternative approach. Next, via a location-limited channel (they used IrDA), a secret was exchanged between the user and the printer. Thus, the user and the printer were able to authenticate and confidentially exchange information through a wireless radio frequency channel. The authentication protocols were based on public key cryptography or the Guy Fawkes protocol [35]. In addition, group key exchange protocols were given to enable secure group ad hoc communications. Since significant human effort was required, such as identifying a printer, exchanging a secret via an IrDA channel, and detecting attacks during group key exchange protocols, the usability of the protocol may be limited. The Resurrecting Duckling security policy and Balfanz's work did not solve the problem of determining whether to trust an unfamiliar service within public environments? In other words, an unfamiliar service might be a malicious service.

Zakiuddin, et al. noted a few authentication issues in pervasive computing environments in their position paper [36]. Entity names were not important because there were too many devices and a name itself did not represent much information. Location information, device types, and states were important attributes.

Stajano and Anderson respectively present their view of security in pervasive computing in their books [2, 37]. Anderson's book is more on engineering side and many of the contents are from his experience in commercial projects. While Stajano's book discusses authentication, integrity, confidentiality, availability, and anonymity. Among these topics, current status about availability and anonymity are pretty good.

A proxy-based approach is one way to assist mobile applications. Burnside, et al. had another proxy-based solution for secure service discovery to enable low processing power devices in infrastructure environments discover each other [38]. Langheinrich designed a proxy-based privacy-aware system [39].

Zhang and Kindberg, at Hewlett-Packard Labs, presented an infrastructure for nomadic users to securely access to a group of distributed services [40]. For example, a user accesses necessary resources for a conference at a hotel. Their model had three phases. In the set-up phase, a central control service, called Lobby, exchange information with all the services securely. Then in the user registration phase, Lobby gave a user some credential. Finally in the user access phase, the user accessed the set of service using the credential. Both public-key and symmetric-key versions were provided.

Based on the notion of a secret set, services only know whether the user was allowed to access the service itself, but not for other services.

Abadi and Fournet proposed two private authentication protocols for ad hoc networks [41]. The protocols enable two principals to authenticate and establish a secure communication channel without explicitly specifying their identities. Assuming that the public keys of the principals are known, authentication messages are encrypted using the other principals' public keys, and thus only the one that holds the private key understands the messages. Unlike protocols in which the target principal is assumed to be known in the vicinity, a target lock in our case needs to be discovered. Moreover, their protocols do not offer protection from malicious insiders. A principal either exposes or refuses to expose, and therefore they are not suitable to discover locks. The Master Key uses different exposure strategies to protect privacy information from insiders, unless it is necessary to expose.

2.7. Entity Authentication in Pervasive Computing

Magnetic stripe cards were first used for the London Underground Transit Systems in early 1960s [42]. Magnetic stripe technology is now widely used as access tokens. For example, hotel guestroom locks and employee badges use the technology, while the most common usage is for bankcards. Most magnetic stripe cards contain three tracks on which data or even PIN numbers are encoded and stored. Since magnetic stripe cards are relatively easy to forge, the annual loss in the UK alone due to counterfeit cards is about £130 million in 2004 [43].

The first smart card was developed in the late 1960s [44], while wide adoption has only recently occurred. Many smart cards contain not only memory but also microprocessors and operating systems. The chips on the smart cards also have tamper-resistant features. Due to the smart cards' computational and storage capabilities and better security features, they are considered the replacement for many magnetic stripe card applications. Moreover, smart cards are also used as prepaid transit cards, ID cards, health cards, or are even embedded within passports [45]. Smart cards may be classified as contact or contactless cards. The former type of cards needs physical contact with readers, while the latter conveniently works over wireless links. A smart card may support multiple applications or authentication purposes.

RFID tags (i.e., passive and active tags) are used as authentication tokens, such as in [46, 47], respectively. However, RFID tags can be tracked by reading tag IDs and thus they are not appropriate for entity authentication when privacy is a concern. Several solutions are proposed to improve privacy for RFID tags in some situations [48, 49], but the privacy problem is not entirely solved. Moreover, passive RFID tags do not have processing capabilities to perform cryptographic operations. The MIT Card System uses magnetic stripe and passive RFID technologies. The usage of passive RFID tags on the cards introduces several new vulnerabilities [50], which makes the cards even less secure than their old magnetic stripe ID cards.

Remote Keyless Entry systems are commonly installed on new automobiles and garage-doors. On a typical Remote Keyless Entry system [51], when its owner pushes a button, the remote control sends a message (8 or 16 bytes) to the receiver. The message contains a "rolling code" for authentication [52]. The "rolling code" is a pseudo-random number that is generated both at the controller and the receiver by using the same seed. The seed is computationally difficult to find from the pseudo-random numbers.

Table 2-4. Comparison of entity authentication technologies.

	Built-in processing power	Memory size	Power source	Authentication	Tamper resistant	Multiple authentication support	User interface
Magnetic stripe	None	Hundreds of bytes	N/A	One-way	No	No	No
Smart card	Yes (some cards)	Up to 8KB	From readers	Mutual	Yes	Yes	No
RFID (passive)	Limited capability	Dozens of bytes	From readers	One-way	No	No	No
RFID (active)	Yes (some tags)	Up to 1MB	Battery	Unknown	No	No	No
Remote Keyless Entry	Limited capability	Limited (no memory module)	Battery	One-way (typical) or mutual	No	No	Buttons
iButton	Yes (some iButtons)	Up to 134KB	From readers and battery (some)	Mutual	Yes	Yes	No

iButtons are used as keys, e-cash, and asset management devices. For instance, in New York City over 200,000 iButton owners are using iButtons to access over 10,000 buildings [53]. Interestingly, memory and processor chips in the iButtons are encapsulated within stainless steel cans. The cans serve as the iButton's network interface when iButtons touch readers. Some iButtons support password protected memory data, challenge-response authentication, or even public key encryptions. Moreover, large memories on iButtons allow an owner to store many keys and certificates for various applications. In Table 2-4, we compare the devices that we have discussed.

Location-based or proximity-based authentication also inspires our work. Improving usability is a major goal of these approaches. For example, the Active Badge, a pioneer location sensing system, may be used as a remote control to start a user's X window display on the nearest computer, and then a user pushes buttons on the badge to control the display [54]. Based on relative location context, content of the applications and displays may be adapted. When a user leaves, applications and the display are automatically closed. With location or even orientation information, the Master Key design could be further improved and unnecessary computation and communication overhead will be reduced.

In another example, Zero-Interaction Authentication (ZIA), an owner wears a token to secure files on his laptop without actively typing the password [55]. When the owner leaves, all file systems on the laptop are protected via encryption. When detecting that the owner returns (i.e., the token is nearby), the laptop fetches decryption keys from the token and restores itself to the state before the owner left. Since there is no owner's involvement in an authentication, ZIA achieves optimal usability. XyLoc uses a similar idea for automatically locking and unlocking a PC [56]. However, these approaches may not be suitable for applications that are beyond trustworthy locks and key owners. If an owner wears tokens for all his authentication tasks, a malicious insider (a lock or other key owner) might query a token to track the owner without the owner's knowledge.

Beaufour and Bonnet proposed to use Personal Servers as digital keys [57]. In their design, a lock actively looks for devices to make a connection. Once a connection is established, a lock identifies itself. If the device is a Personal Server with digital keys, it identifies and proves itself to the lock. Otherwise, the lock marks the device as an invalid key device. Sure enough, a lock that initiates an authentication process simplifies the design such that a Personal Server can easily select the key for the lock. Nevertheless, it is irrational for some devices to take the initiative because most processing and communication efforts are wasted. Such waste on a car's Remote Keyless Entry system, for example, drains the same battery that the car uses to start the engine. If a car is parked for several weeks, the battery in the car may become completely drained. Moreover, without mutual authentication and privacy protection, malicious attackers may send a fake message to query one's identity and track him. The Master Key takes a different design approach and addresses these security and privacy issues.

Now, we present a taxonomy of existing digital key-lock interaction design. The taxonomy provides good a reference point for us to design the Master Key and identify problems. In Figure 2-2, we illustrate the taxonomy of design characteristics, namely a key's initial status, how keys and locks communicate with each other, and the degree of their exposure. We compare the design choices for different digital keys in Table 2-5.

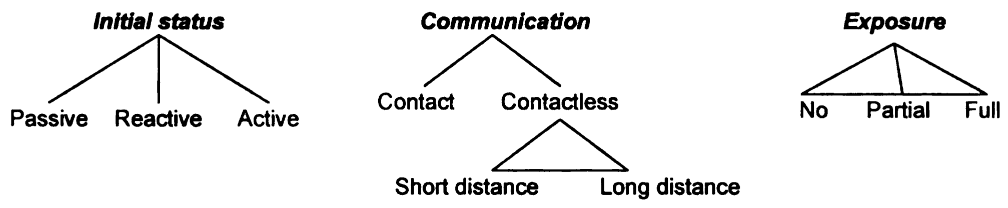

Figure 2-2. Key-lock design characteristics.

A key's initial status may be *passive*, *reactive*, or *active*. If a key is passive, it requires an owner to insert the key into the lock, such as a magnetic stripe card or an iButton. While being reactive, a key is ready to respond upon receiving an authentication message from a lock. For example, when using a passive RFID tag or a contactless smart card as a key, the key is reactive. After receiving a message and power from a lock (reader), the key proves itself to the lock. Active means that a key initiates the authentication process and transmits a message. For instance, in a Remote Keyless Entry system for a car, the key (remote control) becomes active when an owner pushes a button, while the system in the car wakes up frequently (every 20ms) to receive commands from the key [51]. Unlike the passive and active keys, a reactive key does not require its owner to initiate an authentication process. Thus, a reactive key might be maliciously involved in an authentication without its owner's knowledge, or its owner might be tracked via the key. Unfortunately, a key usually does not provide a user interface to notify an owner that an authentication process occurs.

During communication, some keys may physically *contact* locks in order to operate locks. (These keys are usually passive too.) Other keys may operate locks via wireless links and thus are *contactless*. Contactless interaction may work over a span from *short distance* to *long distance*. Contactless smart cards interact with locks within a few centimeters, whereas a key in the Remote Keyless Entry system works over dozens of meters. Contact keys are the least convenient and slowest to use because they need to physically contact locks. Long distance contactless keys do not have the limitation of locks that need to be within the reach of a key owner. During authentication, contact keys implicitly provide strong context that the key is present. Although one might assume a similar context that contactless keys are within the vicinity, such assumption is invalid. For instance, a RFID tag usually is accessible within a short distance, but a malicious attacker may build a powerful reader to read tags [50]. In addition, Mafia fraud attacks are possible. Such attack takes place while contactless keys are not within the vicinity of a lock.

Table 2-5. Comparison of key-lock interaction design.

	Initial status	Communication	Exposure
Magnetic stripe	Passive	Contact	Full, key only
Smart card (contact)	Passive	Contact	Full, some key only, most mutual
Smart card (contactless)	Reactive	Contactless, 10 cm or less	Full, mutual
RFID (passive)	Reactive	Contactless, a few feet	Full, key only
RFID (active)	Active	Contactless, hundreds of feet	Full, key only
Remote Keyless Entry	Active	Contactless, dozens of feet	Full, most key only, some mutual
iButton	Passive	Contact	Full, mutual
ZIA	Reactive	Contactless	Full, mutual
Personal Servers as digital key	Reactive	Contactless	Full, key only

If only a lock verifies a key, it is one-way authentication. The key exposes its information to the lock in *full*, while the lock exposes *no* information to the key. If both a lock and a key verify the other party, it is mutual authentication. Thus, both parties expose their information in full. Potentially, a key and a lock may expose *partial* information and verify the other party's authenticity in multiple rounds. Therefore, a key and a lock can avoid full exposure when it is unnecessary. Such choices are very useful when we design the Master Key in Chapter 7.

CHAPTER 3. SERVICE DISCOVERY IN PUBLIC INFRASTRUCTURE ENVIRONMENTS

3.1. Introduction

Handheld and wearable computers are becoming more powerful and practical. As prices decrease, there are more mobile services and users. Meanwhile, these mobile devices increasingly support nomadic users by offering 2.5G/3G, wireless LAN, or Bluetooth capabilities.

Based on the scenario in Chapter 1 Section 1.1, we identify that security, privacy, and location-awareness are important in service discovery for both nomadic users and services. Splendor considers environments, in which services may be discovered, but mobile users and services may not have accounts in the infrastructure systems. Therefore users, services, and network infrastructure systems are not trusted by each other. To our knowledge, most service discovery protocols are designed for home or enterprise environments, but not for these types of untrustworthy environments, which Splendor targets. We propose a new model to support mobility, security, and user privacy, while in the meantime we integrate location-awareness to service discovery. The security protocols in Splendor enables all parties to mutually authenticate each other, no matter if users have accounts in the network infrastructure systems or not. The security protocols also provide service authorization, confidentiality, integrity, and non-repudiation capabilities. Moreover, Splendor integrates location-awareness, which we believe that the integration not only helps the service discovery, but may lessen the service discovery network infrastructure requirements as well. While providing these functionalities, Splendor still keeps the applications as easy to use as possible.

3.2. Architecture

In this section, first we describe how Splendor provides a new model to support mobile clients and services. Next, we illustrate our integration of location-awareness to our service discovery protocol. Then, we show the security support for all service discovery parties. Last, we discuss how Splendor supports privacy.

3.2.1. A New Service Discovery Model

For simple environments, such as home environments, *client-service models* are usually used. There are two types of components: clients and services. Clients look for services and services reply if they match required service attributes. Then clients select services to use. In more complex environments, *client-service-directory models* are deployed. Clients query directories; services register with directories and provide services to clients; directories cache service information and answer clients' queries. After receiving a matched service list from directories, clients select services, contact services, and then start to use services.

Providing security in enterprise environments, servers may be used to store and maintain user information. System users authenticate with the servers and are authorized to use services. For other environments, such as shopping malls, users do not have accounts on the shopping mall's computer systems. Third-party servers may be used for mutual authentication between each communicating pair, client-service, client-directory, and service-directory, as shown in Figure 3-1(a). These third-party servers may be trusted servers, as in SSDS, or untrusted servers. They may be inline, online, or offline third-parties [58]. We use this model to provide security for stable services. One advantage of this model is that the client-service-directory model may be used with little modification to support security for many situations. One disadvantage, however, is that many limited resource mobile services are burdened with various security checks and maintenance themselves.

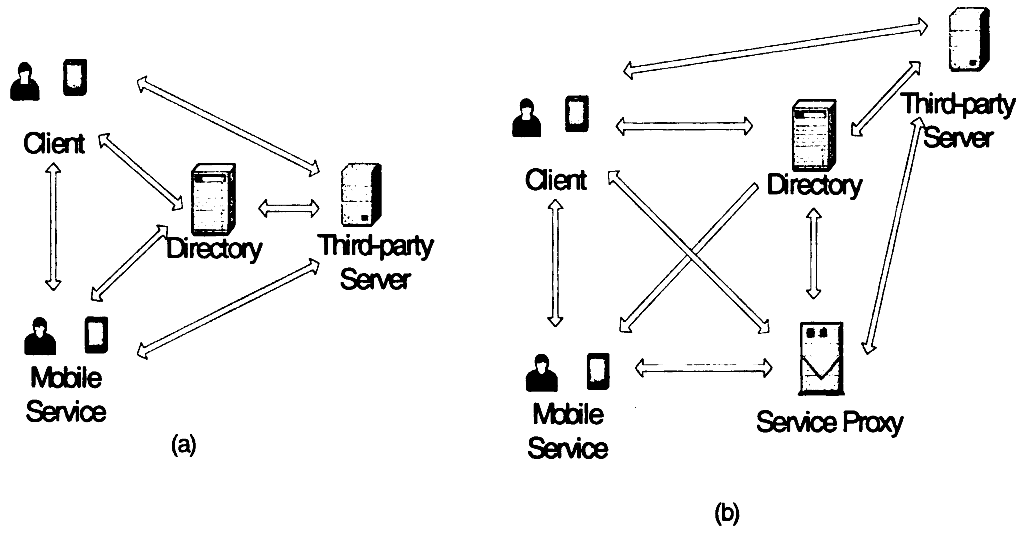

Figure 3-1. Two secure service discovery models. (a). Client-service-directory model with third-party servers. (b). Client-service-directory-proxy model with third-party servers.

In Splendor, we propose a new model that has four types of components: *clients*, *services*, *directories*, and *proxies*, shown in Figure 3-1(b). By including proxies, we may achieve privacy for service providers, offloading much mobile service's computational work to the proxies, and enabling mobile services to do authentication and authorization easily. We focus on this model for the rest of this chapter. Non-mobile services work the same way as in the client-service-directory model, and we do not discuss in detail here.

3.2.2. Tag-based Location-aware Service Discovery

There are many technologies to do location sensing. Let's assume that we have a passive sensing system, which lets mobile clients and services read location information. The sensing system consists of two types of components: readers, which are attached to the mobile clients or services; beacons, which emit information either periodically or after reader's requests.

We use tags to label locations and people, for example, entrances of shopping malls and stores, or Patrick's and David's clothes. Tags, which label places, emit location information and optionally directory's addresses and the directory's certificates. (We use X.509 public key certificates [58] in Splendor, which are discussed in Section 3.2.4. For shorthand we use the term certificates.) The clients use the location information to search for the relevant services. Mobile services use the tags' information to determine that they have moved to new locations and notify their proxies. Tags attached to people's belongings other than the PDAs, such as clothes, may be used to verify that the PDAs are still in possession of their owners.

We show a snapshot that uses location tags for service discovery in Figure 3-2. In the upper half of the figure, we show a scenario in which a client may use a service through its 2.5G/3G connection. In the lower half, a client may use a wireless LAN for service discovery. Most service discovery protocols have an assumption that clients, services, and directories are using one underlying network connection for service discovery. We argue that location information not only provides better precision for location dependent service discovery, it also provides more flexible network infrastructure for service discovery. Service discovery may work with available wireless LANs, 2.5G/3G, other network connections, or combinations of these as long as the directory's addresses are known. Client queries, service registration, and client-service interactions do not need to be bound to any network connections.

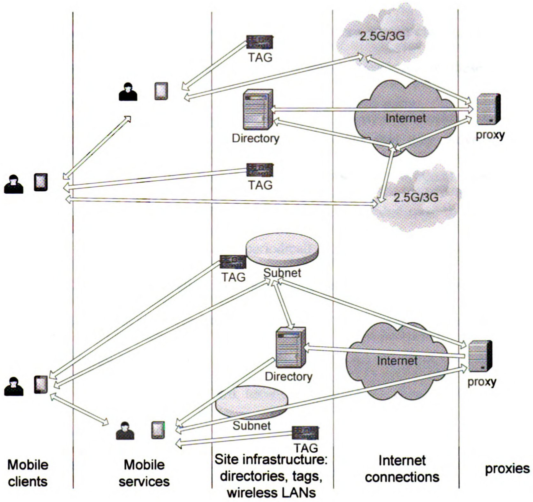

Figure 3-2. Location-aware service discovery architecture with clients, mobile services, directories, and proxies.

3.2.3. Splendor Service Discovery Protocol

In this section, we first discuss the initial steps that clients, services, and directories take before a service discovery. Next, we discuss the announcement and query mechanism of our four-party service discovery. Then we focused on support for mobile services.

3.2.3.1 Bootstrapping

Multicast addresses are used for initial communications among clients, services, and directories. All communicating parties have a priori knowledge of these multicast addresses, so that no manual configuration is necessary for any party.

Mobile clients and services have three ways to tell that they move into new environments. First, they may find that they are attached to different networks, for example, that they acquired new IP addresses or were handed over to new wireless Access Points. Second, directories periodically announce their unicast network addresses, and send out solicitation messages asking services to register. After receiving these messages, mobile clients and services notice that the messages are from directories, which are in charge of other domains. Therefore, there are new directories around and they have moved to new places. Third, location-aware mobile clients and services may read location tags; hence they are able to tell whether they are in new places.

When mobile clients or services move to a new place, they may solicit directories for announcements. Using the directories' unicast addresses, clients may query for services and services know where to register.

3.2.3.2 Service Announcements and Lookups

After bootstrapping, the communications among clients, services, proxies, and directories are all unicast. In this way, only parties, which are necessary to be involved, handle messages. Mobile services authenticate with proxies and ask proxies to handle registration, authentication, authorization, and key management for them. (Mobile services may do all the work themselves without contact proxies.)

Proxies are trusted servers for services. They manage services, for example a medical organization offering licensed physicians for M911 services. When clients query the directories, they may receive some services represented by proxies. Therefore, clients need to contact proxies first and then services. Although an indirection is added into the interactions, removing overwhelming tasks such as service authorization from mobile devices is rewarding. Security policies may be easily modified at the proxy's side without worrying about deployment problems. Mobile service providers do not need to manage those security policies and settings.

Directories cache service information and answer client's queries. They also verify clients' and services' identities. Furthermore, directories may provide support for mobile clients, such as confirmation that the services' authenticities are checked and help to set up connections with services. While announcing their unicast addresses, directories may also announce their public key certificates. Clients and services use the certificates to verify and authenticate directories.

Most service discovery protocols store service states as soft state in directories [59]. Thus, services announce their lifespan. Before the services expire, they announce again. Handheld devices are less stable, since they may have poor wireless connection, or people may just turn them off. Compared to servers (directories and proxies), mobile services are transient. For that reason, Splendor caches soft state information of mobile services in proxies and deals with the instability simply and well. On the other hand, Splendor stores services represented by proxies as hard state in directories. This means that proxies explicitly register and unregister services with directories. To solve possibility of the inconsistent services' state problem, directories explicitly ask proxies about the services' status.

3.2.3.3 Mobility Support Using Aggregation and Filtering

One difficult problem in supporting mobile services is the extremely dynamic property of the services. Services may come and go, or even be shut down and come up again quite often. As a consequence, the directories keep refreshing the services' status – adding and removing entries. We address this problem by extensively using aggregation and filtering at the service registration stage and service lookup stage.

At the service registration stage, a service may not need to register and un-register itself repeatedly. In our scenario for instance, a physician's PDA will not contact the proxy as long as it is within the same shopping mall, although it moves between different stores. Proxies are also doing aggregation and filtering, whenever it is necessary. For instance, when a physician is in a shopping mall and another physician from the same hospital goes to the same shopping mall, instead of registering two services with the shopping mall's directory, the proxy may only register one. On the other hand, the mapping at the proxy's side is changed from one shopping center associated with one physician to one shopping center associated with two physicians. When another physician from the same hospital goes to the shopping mall, there may be still one record in the shopping mall's directory; or when a physician goes home, the record in the shopping mall's directory stays the same. At the service lookup stage, directories may match and/or filter queries first. When requests go to the proxies, based on the requests, the proxies may filter and match some services.

3.2.4. Security Issues

We consider mutual authentication, service authorization, confidentiality, integrity, and non-repudiation for Splendor [60]. Various public key and symmetric key technologies are used to achieve these goals. Because symmetric key encryption is much faster than public key encryption, we use public key techniques for signature and key management, while using symmetric key techniques for data encryption and data integrity. Each party has two sets of public keys: one for encryption and decryption use, another for signing and verification use [58].

Before communication, each communicating pair first sets up a session. In the session set up stage, a new session key is generated and securely transported to the other party using public key technologies. The session key is only known to the pair. It is used in the following session and discarded after the session finishes. On the contrary, those public keys used in session set up stage are used much longer. After communicating parties acquire session keys, communication data are encrypted using the session keys.

If any party wants to record messages for non-repudiation purposes, it may ask the other party to sign the messages using that party's signing private key. Since the signature uses the private key, which is slow, the messages are hashed first and then the hashes are signed. Then the message and the signature are encrypted using the session key. Using this mechanism, the messages are verifiable, even if the session keys are destroyed. The hashes are used for message integrity. The receiving party hashes the original messages and compares with the hashes.

Since clients, services, proxies, and directories may not belong to the same organization, Splendor is based upon Public Key Infrastructure (PKI) technologies, specifically the X.509 strong two way authentication protocol, which uses certificates

[58] for communicating pairs to mutually authenticate each other and exchange keys. PKI enables strangers to exchange public keys securely and its passive infrastructure makes it very simple to use for end users [61]. X.509 two way authentication does key transport and to use public key certificate technology, thus no online trusted servers are needed to set up sessions and share session keys [58]. A public key certificate for each entity includes a serial number, the entity's name, its public key pair, a signature of a certification authority (CA) on the certificate, etc. (Detail certificate structure may be found in [61].) We assume that the public key infrastructures are available. Certificate revocation and trust models of public key infrastructures are important, but are out of the scope of this dissertation. Suppose there are CAs, which sign public keys for clients, directories, proxies, and services. These four parties acquire their certificates before interacting with others. When receiving a pair of new certificates, each party caches them locally. Before using the public keys in the certificates, the certificates are verified first: the signing CA is trusted, the signature is correct, the certificates are in valid time period and not revoked, and they are used for right purpose [61]. For example, a certificate, which a physician uses for his personal financial use, is not valid for his emergency services.

Service authorization is based on privilege levels. Therefore, mobile services only keep several levels. Proxies assign access levels to clients. Security policies are changed and applied at the proxy's side, thus no policy synchronization is necessary at the mobile service's side.

3.2.5. Security Protocols

In this subsection, we show Splendor's security protocols for authentication, confidentiality, integrity, and non-repudiation. As shown in Figure 3-1(b), the pairs of components that communicate are: clients and directories, services and proxies, proxies and directories, clients and proxies, and clients and services. We show directories' announcements and the protocols among these pairs in Figure 3-3.

3.2.5.1 Directory announcements

Using multicast addresses, directories periodically announce messages including their certificates, unicast network addresses, and signatures on the addresses, shown in Figure 3-3(a). Clients verify the certificates before their service lookups. Services forward the messages to proxies.

3.2.5.2 Proxy – Directory

Key transport between a proxy and a directory is similar to the strong two-way authentication [58] with some suggestions given by Menezes, et al [58]. (In (2), the certificates of the directory are not sent to the proxy, since the proxy has the certificates already.) The protocol authenticates both parties and exchanges a session key. A proxy represents a service and registers with a directory. First, the proxy verifies the directory's certificates. Then, it sends its certificates, a message, a session key encrypted using the directory's encryption public key and signs the message and the encrypted session key using its signing private key, as shown in Figure 3-3(b). The directory verifies the proxy's certificates, signature, and the validity of the message. Then it replies with a message encrypted using the session key.

3.2.5.3 Client-Directory

A client checks the directory's certificates when it receives them. If a client inquiries a directory, it sends a query to the directory with its certificates, a secret session key encrypted using the directory's public key, as shown in Figure 3-3(c). The directory verifies the client's certificates and then records the session key. In this step, only the client shows its authenticity, due to the client has already verified the directory's certificates and it implicitly authenticates the directory in the following messages exchange between them.

3.2.5.4 Service - Proxy

The key transport between a service and a proxy may be based on public-key encryption as the techniques we use for key transport between a proxy and a directory. In some situations, a service provider is a user of its proxy and the proxy is trustworthy. To offload tasks from mobile services such as services on PDAs, we provide an alternative solution, which uses symmetric encryption techniques. We use a symmetric key derived from the service provider's password.

When a service moves to a place and wants to register with a directory, it sends a timestamp, a message requesting to register with a directory, the directory's multicast message encrypted using the derived symmetric key shared with its proxy [58], as shown in Figure 3-3(d). The proxy replies it with a message. We may optionally let the service provider type in a password to make sure that he is still in the possession of the PDA.

Notation:
C is a client; D is a directory; P is a proxy; S is a service.
N_D, a directory's unicast address.
$CertE_X$ is an encryption public key certificate of X.
$CertV_X$ is a verification public key certificate of X.
S_X is X's signature using its signing private key.
T_X is the expiration time of this message, which is from X.
M is a message.
K is a session key shared between the sending party and the receiving party.
$P_X(K, Y)$ means Y generates a session key K and encrypts it with its identity using X's encryption public key.
E_{KXY} is an encryption using symmetric key K shared between X and Y.
t_X is a timestamp which X attaches. h(M) is a hash of a message, M.

Let $A = (T_D, D, N_D)$.

D→C: D→S:	$CertE_D, CertV_D, A, S_D(A)$	(1)

(a). A directory's announcement of its unicast address and certificates.

Let $R_P = (E_{KPD}(t_P, D, M), P_D(K_{PD}, P, t_P))$.

P→D:	$CertE_P, CertV_P, R_P$	(1)
P←D:	$E_{KPD}(t_P, M)$	(2)

(b). Key transport between a proxy and a directory.

Let $R_C = (E_{KCD}(t_C, D, M), P_D(K_{CD}, P, t_C))$.

C→D:	$CertE_C, CertV_C, R_C$	(1)

(c). Key transport between a client and a directory.

S→P:	$CertE_D, CertV_D, A, S_D(A), E_{KSP}(P, t_S, M)$	(1)
S←P:	$E_{KSP}(S, t_S, M)$	(2)

(d). A service forwards a directory's multicast message to a proxy. E_K here is a derived key from S's password. A and $S_D(A)$, a directory's multicast message as shown in step (a).

C←P:	$E_{KPC}(P, C, S, K_{CS}, t_P, M)$	(1)
S←P:	$E_{KPS}(P, C, S, K_{CS}, t_P, M)$	(2)

(e). A session key generated at a proxy and transported to a client and a service.

X→Y:	$E_{KXY}(M, t_X)$	(1)

(f). A message encrypted using a session key shared between X and Y.

Figure 3-3. Security protocols among Splendor components.

3.2.5.5 Client-Proxy

After receiving a matched service list from a directory, a client selects and contacts a service or a proxy. If the client contacts a proxy, it verifies the certificates of the proxy that the directory sent along in the matched list. Next, the client and the proxy authenticate, generate, and transport a session key as proxies communicate with directories. Then the proxy checks the access permission and grants a privilege to the client. The proxy also generates a session key to be used between the client and the service as shown in Figure 3-3(e). If the service does not want to use this session key, it may generate a session key itself, which is encrypted using the client's encryption public key.

Thus, we have shown that all the communicating pairs share session keys. After that, all the communication data are encrypted using session keys. For non-repudiation purpose, data are hashed and hashes are signed before encryption. We show these in Figure 3-3(f) and (g).

3.2.6. Privacy Issues

Very few service discovery protocols have considered privacy issues [59]. In SSDS, communication data are encrypted to prevent information from being exposed to eavesdroppers. In Splendor, we also encrypt communication data. In the M911 scenario, the communications are confidential, which not only prevent eavesdropping, but also avoid exposure to the parties that do not need to know. For example, Patrick's medical history is not exposed to the shopping mall computer systems in a medical emergency situation, because the shopping mall system does not know the session key shared between Patrick and David.

Furthermore, we choose to use a passive location sensing system, so only users are aware of their location, and the location sensing system is not aware of its users. Thus, users' location information is kept private until users want to release their positions.

In the Splendor service discovery model, hiding the identity of a service provider is quite easy. We use the M911 scenario as an example. A physician may go to a shopping mall many times, but very few times there are emergency situations. Providing physicians' help in an emergency at the price of exposing their private information to the directories is not ideal. Splendor uses proxies to provide privacy for mobile services. Before registering with directories, proxies may generate names that only make sense to the proxies themselves for the services. In M911, a proxy registers a physician with a directory as "ABC hospital service provider 1001," instead of registering the physician's real ID. Only proxies keep the mappings from the registered service names to the services. Therefore, the directories are unable to know the actual service providers and they are only aware that a service provider is from a known proxy. The proxies, however, are responsible for the services that they represent and register with the directories. Directories may require proxies to sign their messages to assure proxies' responsibility. If proxies do hide the identities of the services, the directories do not reply to clients with matched services, but with proxies instead.

3.3. Performance Analysis and Evaluation

We analyze the overhead of adding proxies, providing privacy support, location-awareness integration, and security protocols in Splendor. First, the overhead of the indirection caused by adding proxies is small. The difference between the client-service-directory model and the client-service-directory-proxy model is that in the former model, all communications are within a few network hops, but for the latter model, a number of messages may be sent over the Internet. Nevertheless, the round trip delay over the Internet is not critical for most of the applications at the service lookup stage. Second, the addition of proxies is transparent to the directories. Clients, however, feel the indirection: they mutually authenticate with proxies, but then communicate with services. If mobile services do not generate session keys themselves, there are the same numbers of mutual authentications in the client-service-directory-proxy model and the client-service-directory model. Third, if there are more than one service matched to a client's request and the proxy does not select a matched service for the client, there is another round of message exchange in which the proxy lets the client pick one service. Fourth, integrating location awareness has overhead for mobile clients and services, but reading location tag information does not have overhead on the critical path of the service discovery processes.

Our software running on PDAs are developed using Microsoft eMbedded Visual C++ 3.0. We measured average overhead of 1000 security operations for mobile clients and services on a ARM SA1110 206 MHz computer (Compaq iPAQ) with 64MB memory running Microsoft PocketPC 3.0. Other software is developed using Microsoft Visual Studio .NET 1.0. We measured average overhead of 1000 security operations for proxies and directories on an Intel Pentium III 866MHz computer with 192MB memory running Windows 2000 Professional. The cryptography software packages, which we use, come with the development tools.

As we see in Table 3-1, these security operations are fast. The longest operations are the public key operations, which take hundreds of milliseconds. Public key pairs are generated once and used for a long time. Session keys are generated and encrypted using encryption public keys in the session setup stage; the overhead is less than 400 ms. After sessions are setup, messages are exchanged many times, but only symmetric key encryptions are necessary, which take less than less 1ms for 1KB messages. For parties, which need to exchange messages with signatures for the purpose of non-repudiation, the overhead is less than 20ms for 1KB messages. (We choose 1024-bit RSA encryption keys on PC, but 512-bit on iPAQs, because it is limited by the software package that we use for iPAQ.)

Table 3-1. Public key and symmetric key operation overhead.

	PC	iPAQ
Key generation		
RSA encryption public key pair [1]	253ms	395ms
RSA 512-bit signature public key pair	86ms	386ms
DES 64-bit session key	0.03ms	0.18ms
Encryption		
RSA public key encrypting a DES 64 bit session key[2]	247ms	373ms
DES 64-bit symmetric key encrypting a 1K bytes message	0.07ms	0.89ms
Decryption		
RSA 1024-bit private key decrypting a DES 64 bit session key	7.55ms	15ms
DES 64-bit symmetric key decrypting a 1K bytes message	0.07ms	0.82ms
Signature		
Hashing a 1K bytes message and RSA 512-bit signature private key signing	1.42ms	15.5ms
Hashing a 1K bytes message and RSA 512-bit signature public key verification	0.13ms	1.47ms

We have discussed certificate validation in Section 3.2.4. Although we do not discuss PKI here, certificate validation may affect performance. Various trust models and

[1] RSA encryption keys are 1024-bit on PCs and 512-bit on iPAQs. We choose 512-bit, because it is limited by the software package that we use.

certificate revocation mechanisms affect performance differently. Detailed discussion of PKI trust models may be found in [61] and certificate revocation may be found in [61, 62]. We assume that for different applications different trust models may be used. For example, Secure Electronic Transaction (SET) has a hierarchical model for which the certificate validation is efficient [63]. RSA Research tested its CA product on eight million certificates, and the average time of online certificates status checking in the tests is less than 1 second [64]. This result gives us a good estimate of certificate status checking for such a large numbers of certificates.

3.3.1. Formal Verification Using BAN Logic

We formally verify our protocols using BAN logic [65]. It facilitates us to make design decisions and helps us to make protocols more succinct. The advantages of the logic are the freshness and the binding checking.

The BAN logic notation and rules, which we used, are listed in Table 3-2 (a) and (b). More detailed explanation may be found in [65]. Since the logic requires us to explicitly writing down our assumption, we clarify what our protocols' target environments. Table 3-3 shows the assumptions that we use. The first three rows of assumptions are about public key certificates. All parties believe certification authorities; the certification authorities have control over the public key certificates that they generate; and each party believes their own public keys. Next, in the fourth row, each party believes that the timestamps, which it generates, are fresh. The fifth row shows that a party believes a session key it generates; or a party believes a session key it shares with another party before in advance. The last row indicates when receiving a session key from the communicating party, one believes that the other party will generate a good session key.

Table 3-2. BAN logic constructs and rules.

Symbol	Denotation
$A \mid\equiv X$	A believes X
$A \Rightarrow X$	A has jurisdiction over X.
$A \triangleleft X$	A sees X.
$A \mid\sim X$	A said X.
$\#(X)$	X is fresh.
$\{X\}_K$	X is encrypted using key K.
$A \xleftrightarrow{K} B$	A and B share a key K.
$\xrightarrow{K} A$	A has a public key K.

(a). BAN logic constructs.

Rules	
Message-meaning	$\dfrac{A \mid\equiv A \xleftrightarrow{K} B,\ A \triangleleft \{X\}_K}{A \mid\equiv B \mid\sim X}$
	$\dfrac{A \mid\equiv \xrightarrow{K} B,\ A \triangleleft \{X\}_{K^{-1}}}{A \mid\equiv B \mid\sim X}$
Nonce-verification	$\dfrac{A \mid\equiv \#(X),\ A \mid\equiv B \mid\sim X}{A \mid\equiv B \mid\equiv X}$
Jurisdiction	$\dfrac{A \mid\equiv B \Rightarrow X,\ A \mid\equiv B \mid\equiv X}{A \mid\equiv X}$
Other	$\dfrac{A \mid\equiv \xrightarrow{K} A,\ A \triangleleft \{X\}_K}{A \triangleleft X}$

(b). BAN logic Rules.

Table 3-3. Assumptions of the security protocols.

1	$C \mid\equiv \xrightarrow{K_{CA}} CA,\ S \mid\equiv \xrightarrow{K_{CA}} CA,\ D \mid\equiv \xrightarrow{K_{CA}} CA,\ P \mid\equiv \xrightarrow{K_{CA}} CA$
2	$C \mid\equiv CA \Rightarrow \xrightarrow{K_D} D,\ S \mid\equiv CA \Rightarrow \xrightarrow{K_D} D$
3	$D \mid\equiv \xrightarrow{K_D} D,\ P \mid\equiv \xrightarrow{K_P} P,\ C \mid\equiv \xrightarrow{K_C} C$
4	$D \mid\equiv \#(T_D),\ C \mid\equiv \#(t_C),\ P \mid\equiv \#(t_P),\ S \mid\equiv \#(t_S),\ X \mid\equiv \#(t_X)$
5	$S \mid\equiv S \xleftrightarrow{K_{SP}} P,\ P \mid\equiv S \xleftrightarrow{K_{SP}} P,\ P \mid\equiv C \xleftrightarrow{K_{CS}} S,\ P \mid\equiv P \xleftrightarrow{K_{PD}} D,\ C \mid\equiv C \xleftrightarrow{K_{CD}} D,$ $C \mid\equiv C \xleftrightarrow{K_{CP}} P$
6	$D \mid\equiv (P \Rightarrow P \xleftrightarrow{K_{PD}} D),\ D \mid\equiv (C \Rightarrow C \xleftrightarrow{K_{CD}} D),\ C \mid\equiv (P \Rightarrow C \xleftrightarrow{K_{CS}} S),$ $S \mid\equiv (P \Rightarrow C \xleftrightarrow{K_{CS}} S),$ $P \mid\equiv (C \Rightarrow C \xleftrightarrow{K_{CP}} P),\ P \mid\equiv (D \Rightarrow N_D)$

Table 3-4. Idealized protocols and stepwise results.

Part	Sndr/Rcvr	Message	Stepwise results
(a)	D→C: D→S:	$\{\underrightarrow{K}\, D, D\}_{KCA}{}^{-1}$(Encryption), $\{\underrightarrow{K}\, D, D\}_{KCA}{}^{-1}$(Signature), $\{N_D, D, \#(N_D)\}_{KD}{}^{-1}$	$C \models K_D$, $C \models N_D$
(b)	P→D:	$\{\underrightarrow{K}\, P, P\}_{KCA}{}^{-1}$(Encryption), $\{K_{PD}, \#(K_{PD})\}_{KD}$ $\{\underrightarrow{K}\, P, P\}_{KCA}{}^{-1}$(Signature), $\{M, \#(M)\}_{KPD}$	$D \models K_P$, $D \models K_{PD}$, $D \models M$
	D→P:	$\{M, \#(M)\}_{KPD}$	$P \models M$
(c)	C→D:	$\{\underrightarrow{K}\, C, C\}_{KCA}{}^{-1}$(Encryption), $\{K_{CD},$ $\#(K_{CD})\}_{KD}$ $\{\underrightarrow{K}\, C, C\}_{KCA}{}^{-1}$(Signature), $\{M,$ $\#(M)\}_{KCD}$	$D \models K_C$, $D \models K_{CD}$, $D \models M$
(d)	S→P:	$\{\underrightarrow{K}\, D, D\}_{KCA}{}^{-1}$(Encryption), $\{\underrightarrow{K}\, D, D\}_{KCA}{}^{-1}$(Signature), $\{M, \#(M)\}_{KSP}$	$P \models K_D$, $P \models N_D$, $P \models M$
	P→S:	$\{M, \#(M)\}_{KSP}$	$S \models M$
(e)	D→C:	$\{\underrightarrow{K}\, P, P\}_{KCA}{}^{-1}$(Encryption), $\{\underrightarrow{K}\, P, P\}_{KCA}{}^{-1}$(Signature), $\{M, \#(M)\}_{KCD}$	$C \models K_P$, $C \models M$
	C→P:	$\{\underrightarrow{K}\, C, C\}_{KCA}{}^{-1}$(Encryption), $\{K_{CP}, \#(K_{CP})\}_{KP}$ $\{\underrightarrow{K}\, C, C\}_{KCA}{}^{-1}$(Signature), $\{M, \#(M)\}_{KCP}$	$P \models K_C$, $P \models K_{CP}$
	P→C:	$\{C \underleftrightarrow{K_{CS}} S, \#(C \underleftrightarrow{K_{CS}} S)\}_{KCP}$, $\{M, \#(M)\}_{KCP}$	$C \models K_{CS}$, $C \models M$
	P→S:	$\{C \underleftrightarrow{K_{CS}} S, \#(C \underleftrightarrow{K_{CS}} S)\}_{KSP}$, $\{M, \#(M)\}_{KSP}$	$S \models K_{CS}$, $S \models M$
(f)	X→Y:	$\{M, \#(M)\}_{KXY}$	$Y \models M$

Table 3-5. Mechanical deduction of the first security protocol step.

Micro steps	Deduction	Notes
1	$\dfrac{C \triangleleft \{K_D\}_{K_{CA}^{-1}}, C \models \xrightarrow{K_{CA}} CA}{C \models CA \mid\sim K_D}$	Using message-meaning rule. The client sees the directory's public encryption key.
2	$\dfrac{C \models CA \mid\sim K_D, C \models \#(K_D)}{C \models CA \models K_D}$	Using the nonce-verification rule. The client believes that the certification authority believes the directory's encryption key.
3	$\dfrac{C \models CA \models K_D, C \models CA \mid\Rightarrow K_D)}{C \models K_D}$	Using the jurisdiction rule. The client believes that the public encryption key of the directory is valid.
4	$\dfrac{C \triangleleft \{N_D\}_{KD^{-1}}, C \models K_D}{C \models D \mid\sim N_D}$	Using message-meaning rule. The client sees the directory's network address.
5	$\dfrac{C \models D \mid\sim N_D, C \models \#(N_D)}{C \models D \models N_D}$	Using the nonce-verification rule. The client believes that the directory believes its network address.
6	$\dfrac{C \models D \models N_D, C \models D \mid\Rightarrow N_D)}{C \models N_D}$	Using the jurisdiction rule. The client believes that the directory's network address.

We transform our security protocols into idealized protocols as shown in Table 3-4. As the convention of the idealize protocol in BAN logic, we leave out the clear text information, because it may be modified by an adversary party. The idealized protocols have the same goal as the actual protocols. Each step in the actual protocols (shown in Figure 3-3) is mapped to a step in the idealized protocols.

Based on the assumptions, we deduct mechanically to get our stepwise result as shown in Table 3-4. The deduction itself is lengthy, so we only show an example (step (a) in Figure 3-3) in Table 3-5. The deduction makes us clear what we can achieve. For example, in Micro step 2, a client needs to validate that the public key certificate is still in its valid time period. Furthermore, it also needs to check that the certificate has not yet been revoked. Then, it knows that the public key certificates of the directory are still fresh. Micro steps 3 and 6 are the stepwise results as we shown in Table 3-4.

We do not use the logic to verify step (g) in Figure 3-3, because the logic is design for authentication protocols. (Step (g) is rather standard procedure for non-repudiation purpose, thus we omit the verification)

3.4. Summary

We discussed a new service discovery model, Splendor, which supports users to discover services in public environments. Splendor offers mutual authentication among components; simplifies service authorization; provides communication confidentiality and message integrity; and supports non-repudiation. User privacy, data privacy, and user location privacy are achieved. The security protocols also are designed to protect against attacks such as eavesdroppers and replay attacks. Location-awareness is integrated to our service discovery protocol to support location dependent service better and reduce the requirements of the underlying network infrastructure.

Our approach might be further improved in two aspects. The capability of proxies may be expanded to support service discovery in other pervasive computing environments. In addition, different service authorization strategies may be implemented to support different types of users with different service requirements.

CHAPTER 4. SECURE SERVICE DISCOVERY IN PUBLIC ENVIRONMENTS VIA AD HOC NETWORKS

4.1. Introduction

Accessing unfamiliar services in public environments is becoming more realistic as we move towards pervasive computing environments. PDAs, cell phones, laptops are becoming commodities. When these mobile devices can access public services, computing is enabled everywhere, as we discussed in our scenario in Chapter 1.

There are two basic security problems when accessing unfamiliar services as in the above scenario. How is a trust relationship set up between two parties? How is secure ad hoc wireless communication set up? One common vision for future computing is that everything is connected to the Internet. Public services such as wireless access points or printers are very likely to have Internet connections since the Internet connections enable these devices to be managed remotely. Meanwhile, many mobile devices may have more than one network channel, for example, 3G, IEEE802.11x, and/or Bluetooth. These channels not only enable devices to be connected to the Internet, but also enable them to communicate to other devices in the vicinity via ad hoc mode. Ad hoc mode is more efficient for many communications, such as in the above scenario. By using Internet channels, we may facilitate ad hoc communications in order to achieve inexpensive, fast, and secure communications. Unlike existing solution attempts, which seek to use pure ad hoc environments, we shift from pure secure ad hoc communication problems to secure ad hoc communications with assistance from other network connections. Our models are also designed to defend against many attacks, including attacks from malicious services. Moreover, based on service discovery protocols, our framework provides better usability.

4.2. System Design

In this section, we discuss many possible threats and attacks, which we take into consideration when we design our models. Based on our service discovery protocol, we illustrate our communication models, security protocols, and system architecture.

4.2.1. Threats and Attacks

Securely accessing unfamiliar services in public environments is more challenging than conventional service accesses. Public services may not have and maintain user information and users do not have accounts for service accesses. Unfamiliar services might be dishonest or even malicious. Furthermore, users might take "free rides" or even "break" the services.

We consider the threats of disclosure, integrity, and denial of service (DOS) [66]. It is easy for services to detect DOS. If someone jams the wireless channel, the attack may be reported through other network links. If a user abuses a resource, there is no need to do anything as long as there is a service charge for access. In order to protect against eavesdroppers, we use cryptographic technology to encrypt messages. However, it becomes trickier if there are fake services, which allure users and collect users' information, but do not actually provide services. Even normal services may record user information.

For active attacks, we consider the man-in-the-middle attack and the message replay attack. Furthermore, we consider situations when unfamiliar services or service providers may initiate attacks on users. Meanwhile we also consider cases when malicious users attack services. We describe more details of how we protect against threats and attacks when we discuss our communication models.

4.2.2. Secure Ad hoc Service Discovery

We only consider service discovery in single hop ad hoc networks in this chapter. As discussed in Section 2.3, the announcement-query and cache-broadcast approaches represent two methods for service discovery within ad hoc environments. When there is more than one service provider in a public environment, it is more reasonable to use the announcement-query approach since there is no incentive for services to broadcast service information for other service providers. However, it is more efficient for services, which are from the same service provider, to broadcast in turn and share the load of broadcast. Services from the same service provider (sibling services) cache all service information and take turns to broadcast their knowledge of available services. As long as a sibling service broadcasts a service announcement message, which contains correct information of its service, the service will not broadcast again itself. We recommend that the rate of the service announcement should be kept at a low frequency; otherwise services from different service providers might compete against each other and jam the wireless channel by sending out service announcements.

Instead of learning the available services by listening to service announcements, a client may send query messages. Comparing the attributes in the query with its own service attributes, only the matched services reply to the client. A client may also search for all the available services in the vicinity by sending a wildcard query. If more than one service from the same service provider matched the query, then the service that last announced replies with a message that contains the set of matched services. Then a client or user picks a service to use. Each service's state is a soft state. In other words, each service has a life span and will be invalid after its lifespan. To continue providing its service, a service announces its new lifespan before the service expires.

4.2.3. Communication Models

While a mobile client wants to use an unfamiliar service, it is difficult to exchange a key securely in ad hoc environments via a wireless radio frequency channel. Eavesdroppers may learn keys. Likewise, it is difficult to prevent the man-in-the-middle attack. Physical contact or using location-limited channels are two approaches to exchange a key securely. However, the usability decreases, because users need to learn to use these approaches and different devices may have different interfaces. We suggest using a proxy-based approach, which not only facilitates authentication, but simplifies usage as well.

There are four parties in our communication models: mobile clients, user proxies, services, and service providers as shown in Figure 4-1. We assume that services have wireless ad hoc communication channels, via which mobile clients in the vicinity may access the services. They also have Internet connections, so service providers may manage the services remotely. A user's mobile device may have more than a wireless LAN capability, for example the device also has a 3G connection. To access an unfamiliar service, we have two models – one uses 3G channels and one does not. We show the two models in Figure 4-1 and discuss them in detail shortly.

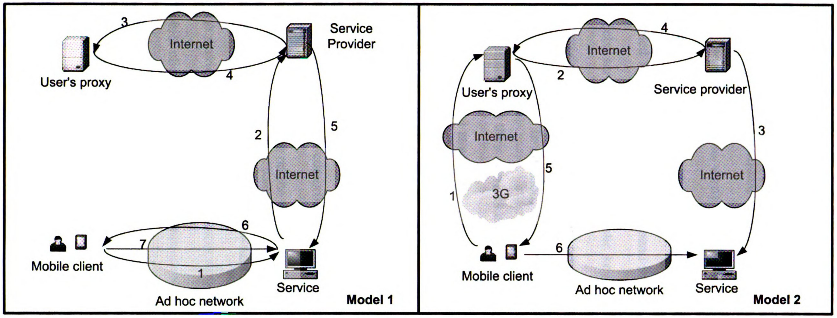

Figure 4-1. Two secure communication models to set up secure ad hoc communications between clients and services.

Both models use user proxies to assist mobile users. The user proxy is a program running on a machine, which is connected to the Internet. The user proxy is designed to fulfill the following functions, which are difficult to achieve in pure ad hoc environments. First, there is a need to verify that a service is the service that it claims to be. Our approach is based on public key cryptography and Public Key Infrastructure (PKI). Detailed PKI information may be found in [61], which we do not discuss here. In our models, service providers have public key certificates, but services do not have them. The user's proxy checks whether the service provider's certificate is valid and used for the right purpose. In addition, the service providers assure the mobile clients and the users' proxies that their services are honest. By using the certificate, we defend against the chosen protocol attack [2]. Second, the user's proxy is used as a safe guard. Every service request from a mobile client goes through its proxy. In case a mobile device is lost, the user may disable the mobile device to access any public services via the proxy. Likewise, if a mobile device's encryption key, which is shared between the mobile client and the user's proxy, is compromised and used for services by a hacker, we may discover it by examining the log on the proxy. Thus, the revocation of the key is simple. Third, more complicated service negotiations such as TrustBuilder may be deployed [67].

A service provider manages services and handles service authorization. Since all the mobile clients use the service temporarily, service authorization is lease-based and only valid for a certain time period. The service provider sends a ticket to the mobile client and a copy to the service. Therefore, services only need to handle a few service access levels. As long as a client's ticket matches the service's copy, access is granted. The service providers imply its assurance of their services when issuing the tickets. In this way, users have more confidence to use unfamiliar services and the service providers have a simple way to stop tempered services and revoke compromised keys shared with the services.

To simplify the discussion of the two models, we suppose that a client has discovered a desired service to access. We discuss the different procedures of the two models to set up secure ad hoc communication channels based on the availability of a 3G connection of a client device.

4.2.3.1 Model 1: Accessing an Unfamiliar Service without a 3G Connection.

In this model, a mobile client only has a wireless ad hoc communication channel. In order to communicate with its proxy, the mobile client has to use the Internet connection that the service has and sends a message to its proxy via a service and a service provider. We outline the interaction in Model 1 of Figure 4-1. First, a client sends a service request message to a service (step 1). Then the service generates its copy of the service request and sends to the service provider along with the message from the client (step 2). Next, the service provider forwards the message to the user's proxy. Along with the message, the service provider also sends its certificate (step 3). After verifying the service provider's certificate, the user's proxy generates a session key for the mobile client and the service. Afterwards, the proxy puts the session key in two copies: one copy for the mobile client, which also implies that the service provider is sound and the service provider has assured the service; another copy for the service provider within its request for accessing the service. The first copy is encrypted using the key shared between the mobile client and its proxy. The other copy is encrypted using the service provider's public key, which is in the service provider's certificate (step 4). When receiving messages from the user's proxy, the service provider decrypts the session key and re-encrypts it using the key shared between the service and the service provider. Then, it sends the session key and the access authorization to the service (step 5). Last, the service forwards the session key from the user's proxy to the mobile client (step 6). Now the mobile client and the service share a key for secure communication (step 7). We show this protocol in Figure 4-2 (a). (We use a notation similar to the BAN logic notation [65].)

Furthermore, when the mobile client sends a service request to its proxy, the message is encrypted using a key, which is shared between the mobile client and the user's proxy beforehand. This encrypted message protects the user from dishonest services, which might alter the service requests.

One possible attack is that a mobile client does not actually access any services but takes a "free ride" and sends packets to a machine on the Internet via step 1, 2, and 3 of Model 1 in Figure 4-1. To guard against this attack and protect services and service providers, service providers check the address of the destination proxy to prevent free rides.

The drawback of the model is that privacy information of the user and user proxy may be sacrificed. An eavesdropper or a service from a competitive service provider may learn information about users, their proxies, and other services from the interaction. Meanwhile, an extra load is placed on the services and the service providers to forward messages.

Notation:

C is a mobile client; S is a service; P is a service provider; U is a user's proxy.
t_X or t_{XY} is a timestamp, which X attaches.
T_X is the expiration time of the service for a client to access, which X attaches.
$CertE_X$ is an encryption public key certificate of X. $CertV_X$ is a verification public key certificate of X.
K_{XY} is a symmetric encryption key shared between X and Y.
$(\ )_{KXY}$ is an encryption using symmetric key K shared between X and Y.
$(\ )_{KX}$ is an encryption using the public encryption key of X.
$(\ )_{KX}^{-1}$ is X's signature using its signing private key.
G_X is the granted privilege to X. A_{XY} is the access code for X to access Y.
M is a message.

Step	Sender/Receiver	Message
1	C→S:	C, t_{C2}, U, $(S, P, t_C)_{KCU}$
2	S→P:	S, $(C, U, t_S)_{KSP}$, $(S, P, t_C)_{KCU}$
3	P→U:	P, $CertE_P$, t_P, $(S, P, t_C)_{KCU}$
4	U→P:	$(t_P, K_{CS})_{KP}$, $(K_{CS}, t_C)_{KCU}$, $CertV_U$, $(C, S, P, t_P, K_{CS})_{KU}^{-1}$
5	P→S:	$(K_{CS}, t_C)_{KCU}$, $(C, G_C, T_P, t_S, K_{CS})_{KSP}$
6	S→C:	t_{S2}, $(K_{CS}, t_C)_{KCU}$, $(t_{C2}, G_C, T_P)_{KCS}$
7	C→S:	C, $(t_{S2}, M)_{KCS}$

(a). Model 1: accessing an unfamiliar service without a 3G connection.

Step	Sender/Receiver	Message
1	C→U:	C, $CertE_P$, $(S, P, t_C)_{KCU}$
2	U→P:	U, S, $(t_U, K_{CS}, K_{UP})_{KP}$, $CertV_U$, $(U, S, P, t_U, K_{CS}, K_{UP})_{KU}^{-1}$
3	P→S:	$(A_{CS}, K_{CS}, G_C, t_P, T_P)_{KSP}$
4	P→U:	P, $(t_U, A_{CS}, T_P)_{KUP}$
5	U→C:	$(t_C, A_{CS}, K_{CS}, T_P)_{KCU}$
6	C→S:	$(A_{CS}, M)_{KCS}$

(b). Model 2: accessing an unfamiliar service with a 3G connection.

Figure 4-2. Security protocols of the two models.

4.2.3.2 Model 2: Accessing an Unfamiliar Service with a 3G Connection.

In this model, the client's mobile device also has a 3G connection. Instead of communicating through the service, the mobile client directly contacts its proxy via a 3G connection. This model provides a more efficient way of communication than Model 1, while incurring the price of the 3G-connection cost. However, the cost is very low: only two messages are required for each service access. Additionally, it is more difficult for eavesdroppers to listen to the 3G and wireless ad hoc channels at the same time (the 3G connection is encrypted [2]).

In the service discovery process, a mobile client learns a service provider's certificates. Along with its service request messages, the client also forwards the certificates to its proxy (step 1 in Model 2 of Figure 4-1). After verifying the certificates, the user's proxy contacts the service provider (step 2). If the access is granted, the service provider sends an authorization message to the service first (step 3) and then sends the message to the user's proxy (step 4). Last, the user's proxy forwards the session key, which is used between the mobile client and the service (step 5). Thus, the mobile client is ready to access the service (step 6). We show the interaction of Model 2 in Figure 4-2 (b).

In comparison to Model 1, access control is simplified. The service provider does not differentiate which mobile client accesses the service, but only records which user's proxy asks for the service. The service provider generates a service access code. Using the service access code, a client obtains the right to use the service.

4.2.4. Building Centralized User Proxy Farms

The user proxies in the two models that we discussed are distributed. An alternative approach is to have centralized user proxies, called a *user proxy farm*. A user proxy farm runs many copies of the user proxy programs, each of which represents a user. We show the architecture of using user proxy farm in Figure 4-3. The idea of aggregating proxies together and run on a machine may be found in [38].

Both models that we discussed in Section 4.2.3 works with the user proxy farm approach. A straightforward approach is running multiple copies of the user proxy program and each copy listening on different port. Therefore, user proxies and mobile clients work exactly the same as we have discussed. A more flexible approach is that based on the user identity, as shown in Figure 4-2 step 1 of the two protocols, a copy of user proxy is dynamically generated. After facilitated a mobile client to verify the trustworthiness of a service, the copy of the user proxy terminates. Both security protocols in Figure 4-2 can be used without modification.

4.3. System Analysis

In this section, we formally analyze our protocols. Then, we compare the two communication models, the centralized user proxy approach, and the distributed user proxy approaches. The comparison is focused on privacy and usability issues.

4.3.1. Formal Verification of the Security Protocols

During the several rounds of design and verification processes, we used BAN logic [65] to verify our security protocols formally and mechanically. It helps us make our protocol succinct and facilitates us to find subtle bugs. Moreover, it enables us to express assumptions more explicitly and to present protocols more clearly.

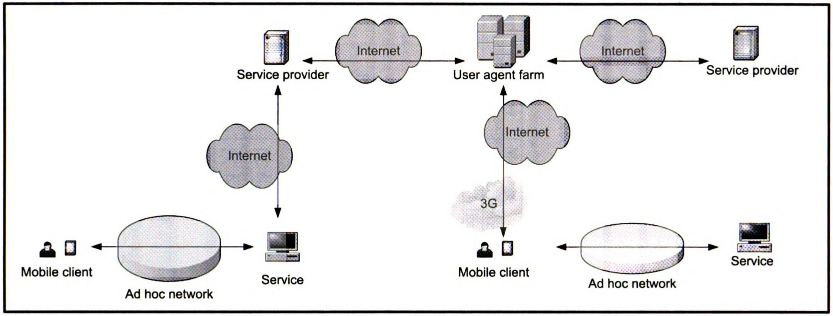

Figure 4-3. An example of a centralized user proxy farm.

The BAN logic notation and rules, which we used, are listed in Table 3-2 (a) and (b). More detailed explanation may be found in [65]. Our idealized protocol of Model 1 is shown in Table 4-1. Each step in the actual protocol (shown in Figure 4-2 (a)) is mapped to a step in the idealized protocol. As the convention of the idealize protocol in BAN logic, we leave out the clear text information, because it may be modified by an adversary party. The idealized protocol has the same goal as the actual protocol.

Table 4-1. Idealize protocol of Model 1. (CSPX is a service request message, which X generates.)

Step	Sndr/Rcvr	Message
1	$C \rightarrow S$:	$\{CSP_C, t_C\}_{KCU}$
2	$S \rightarrow P$:	$\{CSP_S, t_S\}_{KSP}$, $\{CSP_C, t_C\}_{KCU}$
3	$P \rightarrow U$:	$\{\underrightarrow{K}\,P, P\}_{KCA^{-1}}$, $\{CSP_C, t_C\}_{KCU}$
4	$U \rightarrow P$:	$\{C \underleftrightarrow{K_{CS}} S, \#(C \underleftrightarrow{K_{CS}} S)\}_{KP}$, $\{C \underleftrightarrow{K_{CS}} S, \#(C \underleftrightarrow{K_{CS}} S)\}_{KCU}$, $\{CSP_U, \#(CSP_U), C \underleftrightarrow{K_{CS}} S\}_{KU^{-1}}$, $\{\underrightarrow{K}\,U, U\}_{KCA^{-1}}$
5	$P \rightarrow S$:	$\{C \underleftrightarrow{K_{CS}} S, \#(C \underleftrightarrow{K_{CS}} S), G_C, \#G_C\}_{KSP}$, $\{C \underleftrightarrow{K_{CS}} S, \#(C \underleftrightarrow{K_{CS}} S)\}_{KCU}$,
6	$S \rightarrow C$:	$\{C \underleftrightarrow{K_{CS}} S, \#(C \underleftrightarrow{K_{CS}} S)\}_{KCS}$ from S, $\{C \underleftrightarrow{K_{CS}} S, \#(C \underleftrightarrow{K_{CS}} S)\}_{KCU}$
7	$C \rightarrow S$:	$\{C \underleftrightarrow{K_{CS}} S, \#(C \underleftrightarrow{K_{CS}} S)\}_{KCS}$ from C

We present our assumptions about the protocols in Table 4-2. The first row states that the communication pairs trust their shared keys, while the second row declares the trust of the public keys. In row 3, we list the assumptions that a party trusts another party, who has control over key creation or message creation. The suspicious assumptions are that a service and its provider believe that a user's proxy will correctly create a session key for its client and the service. From the service provider's point of view, as long as the user's proxy signs or pays for the transaction, it believes that the user's proxy will generate good keys.

An alternative approach is that the service provider creates the session key, but this will introduce another two messages between the service provider and the user's proxy. The last row of the assumptions is about using timestamps as fresh nonce. Synchronized clocks are required between a mobile client and its proxy and between a service and its provider.

Now, we are ready to deduct from assumptions to conclusions. The deduction itself is lengthy. Thus, we only discuss intermediate results or the conclusion after each step as shown in Table 4-3. After step 1, a service sees an encrypted message from a client to its proxy, but the service is not able to see the content. Then after the second step, by using the message-mean, nonce-verification, and jurisdiction rules, a service provider believes that a client requests to access its service. We also base our deduction on the assumption that the request, which the service provider sees, is a fresh service request from its service (not a replay message). As we have discussed above, synchronized clocks are required in this step. The service provider also sees the encrypted message forwarded from the service. During the processing of messages in step 3, the user's proxy validates the service provider's certificate. (We omit the details of certificate verification, and suppose that the certificate is confirmed to be correct.) Meanwhile, based on a similar deduction as we discussed in the process of step 2, the proxy believes that the client requests a service access.

Table 4-2. Assumptions of Model 1

1	$C \models C \xleftrightarrow{K_{CU}} U$, $U \models C \xleftrightarrow{K_{CU}} U$, $U \models C \xleftrightarrow{K_{CS}} S$, $S \models S \xleftrightarrow{K_{SP}} P$, $P \models S \xleftrightarrow{K_{SP}} P$
2	$P \models \xrightarrow{K_P} P$, $U \models \xrightarrow{K_U} U$, $P \models \xrightarrow{K_{CA}} CA$, $U \models \xrightarrow{K_{CA}} CA$, $P \models CA \Rightarrow \xrightarrow{K_U} U$, $U \models CA \Rightarrow \xrightarrow{K_P} P$
3	$C \models (U \Rightarrow C \xleftrightarrow{K_{CS}} S)$, $P \models (U \Rightarrow C \xleftrightarrow{K_{CS}} S)$, $S \models (U \Rightarrow C \xleftrightarrow{K_{CS}} S)$, $S \models (P \Rightarrow U \models C \xleftrightarrow{K_{CS}} S)$, $U \models (C \Rightarrow CSP_C)$, $P \models (S \Rightarrow CSP_S)$, $P \models (U \Rightarrow CSP_U)$, $S \models (P \Rightarrow G_C)$
4	$C \models \#(t_C)$, $C \models \#(t_{C2})$, $S \models \#(t_S)$, $S \models \#(t_{S2})$, $P \models \#(t_P)$, $P \models \#(T_P)$

Several deductions are needed after the service provider receives the messages in step 4. First, since the service provider possesses its private key, it sees the session key for the client and the service from the user's proxy (other rule). Next, the service provider verifies the proxy's certificate and validates the signature of the proxy. Last, we repeatedly use message-meaning for public key, nonce-verification, and jurisdiction rules along with the assumption that the user's proxy creates the session key correctly, we derive that the service provider believes the session key. The service provider also forwards an encrypted message from the user's proxy to the client. After step 5, the service learns the session key from the service provider's message. Meanwhile, it sees an encrypted message for the client. Afterward, from the sixth step, the client not only gets its copy of the session key, but also learns that the service believes the session key. Finally after step 7, the client starts to use the service. It is therefore straightforward that the service believes that the client believes the session key. In summary, we come to a strong conclusion that the client and the service believe the session respectively and believe that each other believes the session key respectively.

Table 4-3. Results after each step and the final conclusion of Model 1.

After step 1	$S \triangleleft \{CSP_C, t_C\}_{KCU}$
After step 2	$P \triangleleft \{CSP_C, t_C\}_{KCU}$, $P \models CSP_S$
After step 3	$U \models CSP_C$, $U \models \xrightarrow{K_P} P$
After step 4	$P \triangleleft \{ C \xleftrightarrow{K_{CS}} S, \#(C \xleftrightarrow{K_{CS}} S)\}_{KCU}$, $P \models \xrightarrow{K_U} U$, $P \models CSP_U$, $P \models C \xleftrightarrow{K_{CS}} S$
After step 5	$S \models C \xleftrightarrow{K_{CS}} S$, $S \triangleleft \{ C \xleftrightarrow{K_{CS}} S, \#(C \xleftrightarrow{K_{CS}} S)\}_{KCU}$, $S \models G_C$
After step 6	$C \models C \xleftrightarrow{K_{CS}} S$, $C \models S \models C \xleftrightarrow{K_{CS}} S$
After step 7	$S \models C \models C \xleftrightarrow{K_{CS}} S$
Conclusion	$C \models C \xleftrightarrow{K_{CS}} S$, $C \models S \models C \xleftrightarrow{K_{CS}} S$, $S \models C \xleftrightarrow{K_{CS}} S$, $S \models C \models C \xleftrightarrow{K_{CS}} S$

The formal verification process for Model 2 is similar to Model 1, thus we do not repeat the discussion. Table 4-4 shows the idealized protocol, assumptions, and results after each step. The conclusion seems weaker than Model 1; that is only the service believes that the client believes the session key shared between them. However, after a message that the service sends back to the client, the client will believe that the service also believes the session key.

The verification of the message contents such as service requests, granted privileges, and access codes are not the standard verification focus of the BAN logic. In our protocols these components are important. The powerful binding and freshness checking capability of the BAN logic may easily extended and used to check the correctness of these components. Based on the same set of constructs and rules, we verified them.

4.3.2. Comparison between the Approaches

The distributed user proxy approach has better privacy then the user proxy farm approach. In the user proxy farm approach, each user's activity is exposed to a central place. A user proxy farm knows the places, the time, and the services that a user used, while the distributed approach keeps all these privacy information. The privacy exposure in the user proxy farm approach has been seen in real life as people use credit cards for purchases. Nevertheless, the more information exposed the more inference may be deducted and thus the less privacy we have.

Comparing the usability, the user proxy farms approach simplifies the management tasks for users. The burden of managing user proxies and maintaining a machine connected to Internet is aggregated at a central place. A third party may provide the service and staff with special skill maintain the user proxies. Many users might be willing to sacrifice the privacy whenever managing machines and software are tedious or overwhelming for them. Further discussion of privacy and usability is out of the scope of this dissertation. A compromise approach might be that each user runs its own user proxy and maintain the service usage records, while outsourcing the management to a third party.

Table 4-4. Formal verification of Model 2 using BAN logic.

Step	Sndr/Rcvr	Message
1	$C \rightarrow U$:	$\{CSP_C, t_C\}_{KCU}, \{ \xrightarrow{K} P, P\}_{KCA^{-1}}$
2	$U \rightarrow P$:	$\{C \xleftrightarrow{K_{CS}} S, \#(C \xleftrightarrow{K_{CS}} S), U \xleftrightarrow{K_{UP}} P, \#(U \xleftrightarrow{K_{UP}} P)\}\}_{KP}$, $\{ \xrightarrow{K} U, U\}_{KCA^{-1}}, \{CSP_U, \#(CSP_U), C \xleftrightarrow{K_{CS}} S, U \xleftrightarrow{K_{UP}} P \}_{KU^{-1}}$
3	$P \rightarrow S$:	$\{ C \xleftrightarrow{K_{CS}} S, \#(C \xleftrightarrow{K_{CS}} S), G_C, \#G_C, A_{CS}, \# A_{CS} \}_{KSP}$
4	$P \rightarrow U$:	$\{ A_{CS}, \# A_{CS} \}_{KUP}$
5	$U \rightarrow C$:	$\{ C \xleftrightarrow{K_{CS}} S, \#(C \xleftrightarrow{K_{CS}} S), A_{CS}, \# A_{CS} \}_{KCU}$,
6	$C \rightarrow S$:	$\{ C \xleftrightarrow{K_{CS}} S, \#(C \xleftrightarrow{K_{CS}} S)\}_{KCS}$ from C

(a). Idealized protocol of Model 2.

$C \models C \xleftrightarrow{K_{CU}} U$, $U \models C \xleftrightarrow{K_{CU}} U$, $U \models C \xleftrightarrow{K_{CS}} S$, $S \models S \xleftrightarrow{K_{SP}} P$, $P \models S \xleftrightarrow{K_{SP}} P$, $U \models U \xleftrightarrow{K_{UP}} P$
$P \models \xrightarrow{K_P} P$, $U \models \xrightarrow{K_U} U$, $P \models \xrightarrow{K_{CA}} CA$, $U \models \xrightarrow{K_{CA}} CA$, $P \models CA \Rightarrow \xrightarrow{K_U} U$, $U \models CA \Rightarrow \xrightarrow{K_P} P$
$C \models (U \Rightarrow C \xleftrightarrow{K_{CS}} S)$, $P \models (U \Rightarrow C \xleftrightarrow{K_{CS}} S)$, $S \models (U \Rightarrow C \xleftrightarrow{K_{CS}} S)$, $U \models (C \Rightarrow CSP_C)$, $S \models (P \Rightarrow U \models C \xleftrightarrow{K_{CS}} S)$, $P \models (S \Rightarrow CSP_S)$, $P \models (U \Rightarrow U \xleftrightarrow{K_{UP}} P)$, $S \models (P \Rightarrow G_C)$, $P \models (U \Rightarrow CSP_U)$, $U \models (P \Rightarrow A_{CS})$, $C \models (U \Rightarrow P \models A_{CS})$, $S \models (P \Rightarrow A_{CS})$
$C \models \#(t_C)$, $U \models \#(t_U)$, $P \models \#(t_P)$, $P \models \#(T_P)$

(b). Assumptions of Model 2.

After step 1	$U \models CSP_C$, $U \models \xrightarrow{K_P} P$
After step 2	$P \models \xrightarrow{K_U} U$, $P \models CSP_U$, $P \models C \xleftrightarrow{K_{CS}} S$, $P \models U \xleftrightarrow{K_{UP}} P$
After step 3	$S \models A_{CS}$, $S \models G_C$, $S \models C \xleftrightarrow{K_{CS}} S$
After step 4	$U \models A_{CS}$
After step 5	$C \models C \xleftrightarrow{K_{CS}} S$, $C \models A_{CS}$
After step 6	$S \models C \models C \xleftrightarrow{K_{CS}} S$
Conclusion	$C \models C \xleftrightarrow{K_{CS}} S$, $S \models C \xleftrightarrow{K_{CS}} S$, $S \models C \models C \xleftrightarrow{K_{CS}} S$

(c). Results after each step and the final conclusion of Model 2.

4.4. Implementation

The two communication models are part of our Splendor project [68], which is a framework for secure, private, and location aware service discovery in public environments. We are prototyping and comparing different model designs in heterogeneous environments. Mobile clients are running Windows CE on PDAs, while user's proxies, service providers, and services are running Window 2000 on PCs or Solaris on workstations. All the messages passing between communicating parties are in XML format, which provides flexibility and simplifies our processing on different platforms.

Services are provided via web interfaces. We run Apache web servers to simulate embedded web servers. Both service administration and service usages are based on simplified privilege levels (no access control list maintained within services.) We also use a wireless LAN to simulate 3G connections, which we care more about the message size instead of the response time.

We have measured encryption algorithms, which are used in the models. The largest overhead of public key operation on an average PC or PDA is less than half second. Most symmetric key operations are a few milliseconds or less. We are making the quantitative comparison of the two models and measuring the scalability of the systems.

4.5. Summary

We presented a proxy-based approach to facilitate ad hoc communication in public environments, based on a service discovery protocol. To access unfamiliar public services securely, we proposed two models. The models utilize existing Internet connections to setup trust relationships and exchange security keys while keeping efficient ad hoc communications. We further discussed and compared the distributed user proxies and centralized user proxy farm approaches. We formally verified and improved our security protocols using BAN logic. Our approach depends on the PKI. Thus, the approach may be limited to users and service providers who have public key certificates.

CHAPTER 5. A PRIVATE, SECURE AND USER-CENTRIC INFORMATION EXPOSURE MODEL FOR SERVICE DISCOVERY PROTOCOLS

5.1. Introduction

Envision that within pervasive environments, dozens to hundreds of devices and services may surround a user. Over time, one may utilize many services at different places. Meanwhile, the user may be the owner of some services. When discovering services in such environments, much information is sensitive and should be exposed with prudence. Unfortunately, most users do not have special skills to manage devices. As we discussed in our scenario in Chapter 1, Bob (as a physician) may not be able to secure devices and network service access. In addition, it is not feasible for Bob to devote much time on system administration.

In this chapter, we present a secure, efficient, and scalable model, called *PrudentExposure*, to allow legitimate users to discover and use services easily, while it excludes others from seeing sensitive information within pervasive computing environments. Our service discovery architecture is user-centric; user identities are managed centrally and are supplied automatically. Hence, users are free from the burden of associating identities with domains and pervasive services. If users have privileges, they do not need a priori knowledge of the existence of services or the knowledge of which domain a service belongs. Moreover, devices are free from authenticating users.

5.2. A New Service Discovery Architecture

Based on the client-service-directory model (see Section 2.3.4), our architecture has four types of components – clients, services, directories, and user agents. Clients and services are similar to their counterparts in the client-service-directory model. A client is a device a user utilizes to access services, for example, Bob's Bluetooth earphone in our scenario. A service is a networked computing resource such as Bob's Bluetooth MP3 player. (A device may change its role, for example, if Bob downloads new songs to his MP3 player from his PC, the MP3 player becomes a client in that context.) Directories store and dynamically update service information. However, they have tight relations with the services (see Section 5.3.1). We introduce a new component, called a user agent, to facilitate users managing identities for authentication.

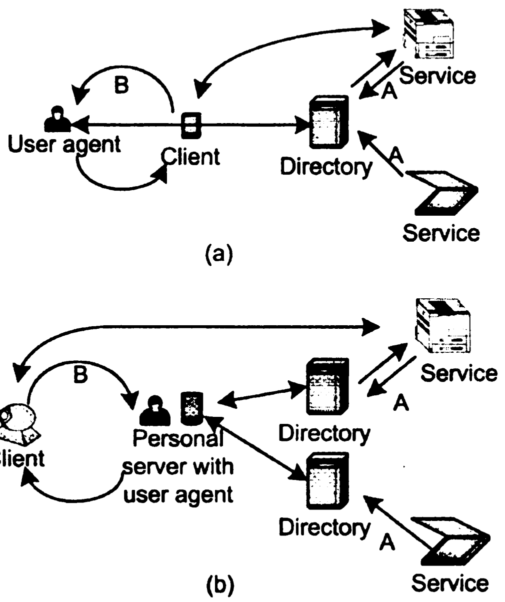

Figure 5-1. Two service discovery examples. (a) A PDA (client) discovering services. (b) A digital camera without a public key operation capability discovering services via a user agent on a Personal Server[69] (a powerful processing and networking handheld device).

The four components are classified as: a service discovering party, which consists of a user agent and a client; and a service provider, which consists of services and a directory.

Two discovery diagrams are shown in Figure 5-1. There are three pairs of relationships among the four types of components: a relationship within a service provider (binding A in Figure 5-1), a relationship within a service discovering party (binding B in Figure 5-1), and a relationship between a service provider and a service discovering party. We discuss the first two relationships in Section 5.3.1 and 5.3.2, respectively. The last relationship is discussed in Section 5.4 when we discuss the detailed discovery process.

5.3. Target Environments

Our model targets pervasive computing environments in which users discover services in their vicinities. Our prototype system uses IEEE 802.11b and UDP multicast, but our model may be deployed to wired and wireless LAN (such as IEEE 802.11x and Bluetooth) or other communication mechanisms that support broadcast or multicast. If a device has multiple interfaces, it may simultaneously communicate via all interfaces to discover services.

In a place, multiple administrative domains may coexist. Service owners (owners in short) or their administrators manage their domains, respectively. For example, in Bob's office, Bob manages his personal devices and services; Alice manages hers; and system administrators manage the office's computing resources. Moreover, users utilize different roles to access services. To access office's services, Bob uses a user name and password pair; to access Alice's services, Bob uses another user name and password pair that he acquires from Alice.

Our model does not require a fixed underlying authentication and authorization mechanism. However, we assume that users have valid identities from service providers.

A user may use a user name and a password to access one service and use a certificate to access another service.

5.3.1. Owner-based Service Management

In insecure service discovery protocols, services announce their information periodically or after directories' solicitations, and directories accept all service registrations. Exposing service information to everyone sacrifices privacy and may cause attacks. In the meantime, directories may be swamped with unrelated service information and degraded their performance. Nevertheless, deploying centralized secure service discovery protocols in pervasive computing environments causes new problems. If a user registers his services with a central directory, he not only exposes his service information to the directory but also loses control of who should acquire the service information. Moreover, in the client-service-directory model, directories announce their existence information periodically or when clients or services discover directories. In pervasive computing environments, if a directory represents a person, the announcements expose the person's presence information.

To solve the problems, service registration in our model is selective and owner-based. A directory only stores information of the services that belong to the same owner. Similarly a service only announces its information to the directory of the same owner. For example, in our scenario, in Bob's office, services owned by Bob register with Bob's directory, and services owned by the office register with the office's directory. Furthermore, Bob's directory does not accept Alice's service registrations. Thus, Bob's services are kept private and service information is exposed under the control of his directory and in turn under his control. In addition, services and directories use soft state and a lease-based service registration mechanism (used in many service discovery protocols [59]) to maintain the freshness of the service information.

In our model, directories do not announce their information periodically. When hearing a discovery message, a directory checks whether the discovery message is sent by a valid user or service. If the check returns a positive answer, the directory replies back. Otherwise, the directory keeps silent. The detail discovery processes are discussed in Section 5.4.

Besides the service registration, a directory and a service have a long-term control relationship: the directory controls the service. This relationship only needs to be set up once. For instance, when the owner first acquires the service, he associates the service with his directory. We borrow the master-slave relationship idea from the Resurrecting Duckling approach [32, 33]. A directory is a mother duck (master) and services are ducklings (slaves). After a duckling (service) accepts an imprinting from a mother duck (directory), a secure communication channel is established. Moreover, only the directory can instruct the service to break this control relationship.

Directories authenticate users and maintain access control lists. All service accesses need to go through the directories to get permissions. When a client is authorized to access a service, a directory informs the service specific policies. Owners (or administrators) can enforce access control and manage user privileges in one place: the directories. By moving the authentication and authorization from services on to the directories, security and privacy requirements on services are dramatically reduced. Managing user accounts and privileges do not need to be done on services individually for heterogeneous devices. Revocation of a user's privilege can be easily done on the directories. Moreover, a domain may be represented by one or more directories. If multiple directories are deployed, user accounts and privileges need to be synchronized.

Using the Resurrecting Duckling approach may be good for a person to manage his personal devices and services. With authentication and authorization aggregated on a device, it becomes easy for a person to keep a directory with him. Other types of control between services and directories may be used, for instance, servers may be used to authenticate and authorize users. Service access requests may occur via directories to the servers for authentication and authorization.

Directories may run on PCs, servers, or portable devices such as cell phones, PDAs, and Personal Servers [69]. These portable devices have reasonably good processing capability, network capability, and storage capacity. In our scenario, a directory may run on Bob's home PC to manage services in his house and another directory may run on his cell phone to manage devices that he carries, in his house, and/or in his office.

5.3.2. User Agent-based Service Discovery

As discussed in the Introduction, it seems infeasible to require clients to support all potential authentication mechanisms in existing service discovery protocols. On the other hand, requiring users to remember services and their domains decreases usability. We use user agents to address the two problems. A user maintains all her identities in a table on her user agent. In the table, a domain identity is associated with a domain, an authentication method, authentication information (such as user name, password, or certificate, etc.), expiration time, and/or the domain's public key. Securing user agents is very important, but it is out of the scope of this dissertation.

Before a service discovery process, a client needs to bind to a user agent. This binding is short-term (one-time) and valid only in the current service discovery process. A secure channel may be established via side channels as discussed in [32], for example, physical touch between two devices becomes as a physical channel. By using these side channels, user agents and clients can exchange session keys and establish secure communication. Moreover, this binding also triggers a service discovery process.

In a service discovery process, a client queries a type of service and a user agent supplies necessary authentication information to gain privileges. The service query result depends on the identities that the user agent supplies. We will discuss our method in Section 5.4. Offloading authentication tasks from clients to user agents simplifies the design of clients, i.e., clients do not need to support various authentication mechanisms. Moreover, it is much easier to add a new authentication mechanism on a user agent than to add it on all clients.

Potential devices to serve as user agents should be handy and available whenever needed. Cell phones, PDAs, Personal Servers and iButtons might be good candidates. iButtons [70] are very small and can be worn as a ring. One type of iButton is able to process various public key operations and is claimed to do a key operation within a second [70]. In our scenario, Bob and Alice may run their user agents on their cell phones, respectively. When Bob uses his Bluetooth earphone to discover MP3 songs, he uses the earphone to touch the cell phone. The touch binds his earphone to his user agent. His user agent supplies identities associated with his user roles and finds songs on his MP3 player and laptop. Similarly, if Alice uses Bob's earphone, the earphone binds to her user agent and in turn finds songs on her MP3 player and PDA. Alice's user agent should not find Bob's MP3 player, unless Bob gives her an identity and grants her privileges.

5.4. The PrudentExposure Model

Before we focus on the interaction between a user agent and a directory, we briefly discuss our PrudentExposure model as shown in Figure 5-2. There are five major steps when discovering a service (Figure 5-2 (a)): domain match, authentication, service selection, key distribution, and invocation. First, a user agent searches available domains. After obtaining the information, it selects the correct identities to authenticate with them. Next, the user agent and the client ask the directories for service information. After receiving replies, the client or the user selects a service. Then, an encryption key is distributed to the selected service and the client. Last, the client is ready to use the service.

There are two steps in a service registration (Figure 5-2 (b)): domain match and registration. In the domain match step, a service discovers available domains. After finding its domain, a service sends an encrypted registration message using the secure channel as we discussed in Section 5.3.1. The two domain match steps in Figure 5-2 (a) and (b) are very similar, thus we will discuss only briefly the domain match step of the service registration in Section 5.4.7.

The domain match step is vital for user agents and directories. Without prudent exposure, the owner's and user's presence information and user identities may be sacrificed. To keep the domain match private and secure, user agents and directories speak *code words*. A user agent says a code word, and then a directory checks whether or not the code word is correct. If the code word is correct, the directory says another code word and the user agent checks. This interaction establishes mutual trust between the user agent and the directory.

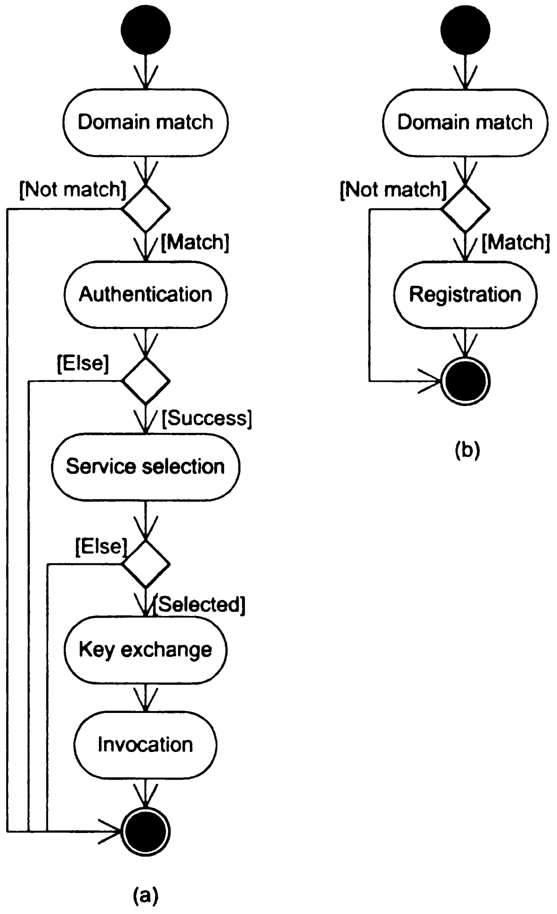

Figure 5-2. The PrudentExposure activity diagrams. (a). The activity diagram of discovering a service. (b). The activity diagram of a service registration.

To address the issues of multiple coexisting domains and discovering at different places, a user agent may say many code words to find the existing domains. We express code words in Bloom filter form [19]. It allows user agents to say many codes words within one network packet. The advantages of using Bloom filters in our case are: security and privacy, simple code word assessments, space efficiency, and scalability.

5.4.1. An Introduction of Bloom Filters

Bloom filters are suggested as an efficient way to test membership [19]. The basic idea has two parts: Bloom filter generation and membership test. To generate a Bloom filter, as shown in Figure 5-3 (a), select several hash functions that have the same range. The Bloom filter is represented as a bit array, whose length equals to the range of the hash functions. Each possible hash result is represented as a bit in the Bloom filter. The filter is initially set to zero. Then for a set of elements, apply every hash function to each element. By using a hash result as an index, we set a bit in the array. Note that a bit may be set many times due to different elements or due to different hash functions. For a membership test of an arbitrary element, the process is similar, as shown in Figure 5-3 (b). First, apply every hash function to the element. Then, use each hash result as an index in the Bloom filter. Any zero found at the index position in the filter means non-membership. Otherwise, the element is a member.

5.4.2. Matching Existing Domains Using Bloom Filters

An owner defines his domain, which is identified by a unique ID, as a *domain identity*. A domain identity is a secret that a domain shares with its users. Each code word exchanged between a user agent and a directory is the hash results of the domain identity. If three hash functions are used, then three bits of the hash results in the filter represent a code word. To simplify the discussion, we suppose only one hash function is used to generate Bloom filters, thus every code word is one bit.

To generate a bit in the filter, first we calculate the hash of the domain identity using the function *h(domain identity)*. Then, we use the mod function, *mod(h(domain identity))*, to determine the bit to set in the filter. (Note *mod(h(domain identity))* is equivalent to the hash function that we have discussed in Section 5.4.1, while *h(domain identity)* is an existing hash function that we utilize. Since we may think the mod function as the last step of a Bloom filter hash function, we do not distinguish them later.) We choose MD5, SHA-1 and RIPEMD-160 as our hash functions because of their good properties of preimage resistance (computationally impossible to find the original message from the hash result) and collision resistance (computationally impossible to find two distinct messages with the same output) [58]. Therefore, given a bit in a Bloom filter, it is computationally difficult to know the original domain identity.

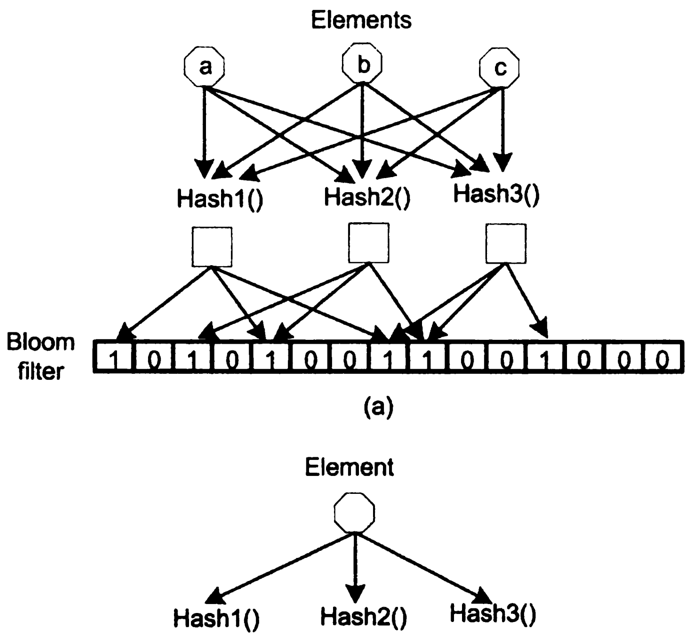

Figure 5-3. A Bloom filter example using 3 hash functions. (a). Generating a Bloom filter. (b) A membership test.

The domain match process in Bloom filter form is shown in Figure 5-4. A user agent generates a Bloom filter by specifying necessary domain identities. Then the Bloom filter is broadcasted to directories for a membership test at the directories. If a directory finds a match, it generates another Bloom filter with two bits set. The first is the bit that the directory finds as a match, and the second bit is set for the same domain identity using another hash function. Then the filter is sent to the user agent for a membership test. It is not necessary to send the entire Bloom filter back to the client because there are only two bits set. A message that indicates which two bits are set is enough. Successfully generating the filters demonstrates the knowledge of the domain identity. User agents and directories agree on the hash functions in advance. User agents use MD5 to generate Bloom filters and directories use it to do membership tests. Similarly, directories use SHA-1 to generate reply messages and user agents use it to do membership tests. The RIPEMD-160 is used as an optional hash function for user agents to decrease the probability of false positive.

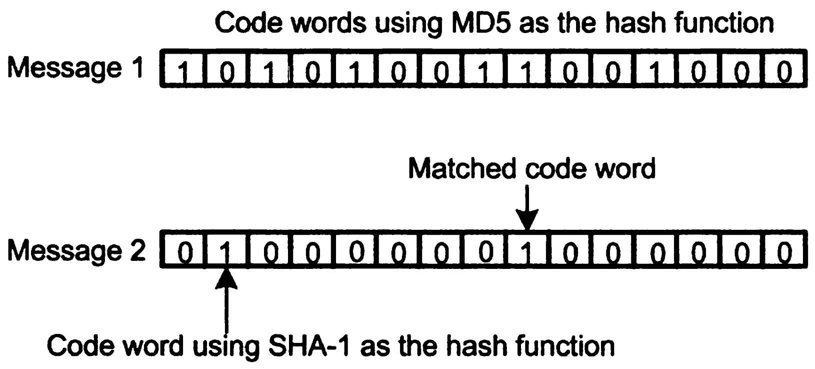

Figure 5-4. Domain match using Bloom filters. A user agent broadcasts Message 1, and then a directory replies with Message 2 when it finds a matched code word.

5.4.3. Exchanging Dynamic Code Words

Replay attacks are possible when exchanging static code words. In our case, an eavesdropper may associate bits with domains without finding the original domain identities. After listening to the domain match messages of a directory, he will find the bit in the Bloom filters associated with the domain by intersecting the filters. Next, he may replay the bit and then check the reply messages to test the existence of the domain. Furthermore, an eavesdropper may physically find out who are the users of a domain by checking the relative bits in the filters. Thus, static code words should be rarely used. However, they may be appropriate for some domains, such as commercial wireless services in airports, where a domain's presence information may not be important. Moreover, the computation overhead to generate dynamic code words is not a concern in most cases, as we show in Section 5.5.4.

To generate dynamic code words, we add a time variant parameter when calling the hash functions, specifically hash-based message authentication codes as discussed in [58]. By adding this parameter, the code word serves as a one-time code word. Therefore, there is no fixed bit in the filters associated with a domain identity and a replay message can be easily detected.

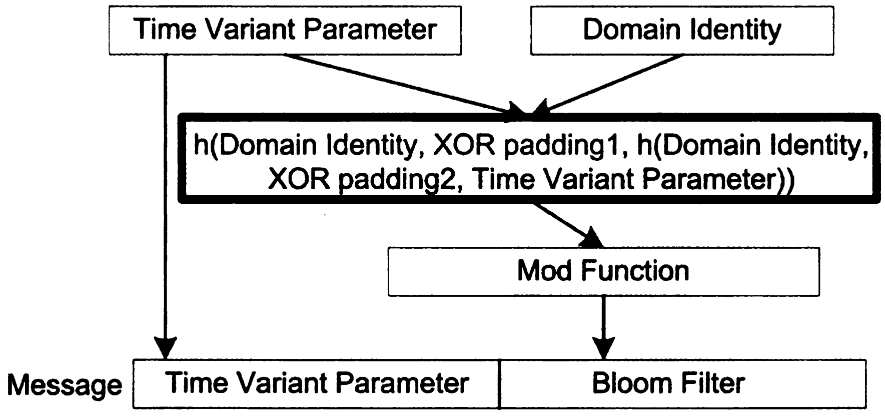
Time Variant Parameter
Domain Identity
h(Domain Identity, XOR padding1, h(Domain Identity, XOR padding2, Time Variant Parameter))
Mod Function
Message
Time Variant Parameter
Bloom Filter

Figure 5-5. Using dynamic information to generate Bloom filters.

The detailed method of setting a Bloom filter bit is highlighted in Figure 5-5 and we use the hash algorithm proposed in [71]. A domain identity is considered a key and the time variant parameter is considered a message. The time variant parameter includes a random number and a timestamp in our case. A user agent generates the time variant parameter and sends it along with the Bloom filter to directories. A directory uses the time variant parameter for the membership test and for the reply message. Moreover, a user agent uses the same time variant parameter for a discovery for all domains.

5.4.4. Preventing Internal Attackers to Act as an Owner

When a user agent finds a match in a reply message, it will try to authenticate. However, a domain identity is a shared secret, and thus a reply message might come from an active internal attacker. If the message does come from an attacker, the attacker is rather sure when he sees the authentication message that a user from the domain is discovering services.

To prevent internal attackers, a directory may sign a reply message with its private key. At the user agent side, after a successful membership test, the user agent may verify the directory's signature using the directory's public key. Since an internal attacker only has the directory's public key, he is not able to sign the message. Thus, he is limited to analyze the Bloom filters as an internal eavesdropper can.

5.4.5. Obscuring Identities to Improve User Privacy

Although an eavesdropper cannot act as an owner, he may still infer that a user is discovering services by analyzing the user's Bloom filter, because every bit set in the filter represents a true code word. If the attacker has some knowledge of a user's code words, the user's presence information may be inferred. For instance, suppose Alice knows that Bob is discovering services and then knows that his discovery message has eighteen bits set. If different employees in the office have a different number of bits set in their filters or if only a few employees set the same number of bits, then a filter with eighteen bits set is very likely sent by Bob. Although a different group of eighteen bits are set in each of Bob's discovery message (because of dynamic code words), the number of bits set may provide a clue that Bob is nearby.

To counter the attacks, a user agent may optionally set more bits in a Bloom filter to mix the true code words with other randomly selected bits. For instance, a user agent may set 80 bits in a Bloom filter, while he has 50 domain identities. Thus, an eavesdropper is unsure how many true code words a user has. To control the false positive rate at the directory side, our default is to set 1 percent of the bits in filters, including the true code words. Thus, query messages look the same.

If a user has many code words, we may use larger Bloom filters and compress them. For example, a user may send an 8K bytes Bloom filter with 650 bits set (1 percent). Such Bloom filters can be compressed to less than 1000 bytes, as our simulation results show in Section 5.5.4.

5.4.6. Protecting Service Request Privacy

A service request specifies a service name and attributes. Nevertheless, it is not necessary to let all directories know the request. In our scenario, Bob may not want to tell the office's directory that he is looking for an MP3 player. Thus, instead of specifying the service name and attributes, a client may ask directories what services are available and authorized for it to access. This is similar to the wildcard search in many service discovery protocols [59]. Unlike those protocols, the reply messages are in the Bloom filter form in our model. The Bloom filters always fit in a single packet, while the message lengths of the existing approaches vary and may not be fit in one packet. Since services and attributes expressed in Bloom filter form were discussed in detail in SSDS

[18], we do not further discuss it here. In short, Bloom filters can express many services and attributes in a filter. In SSDS, hierarchical directories need to handle much more services than in our model, thus we are not concerned about the performance of building and rebuilding the Bloom filters. Unlike SSDS, queries in our model are evaluated at the client side instead of the directory side.

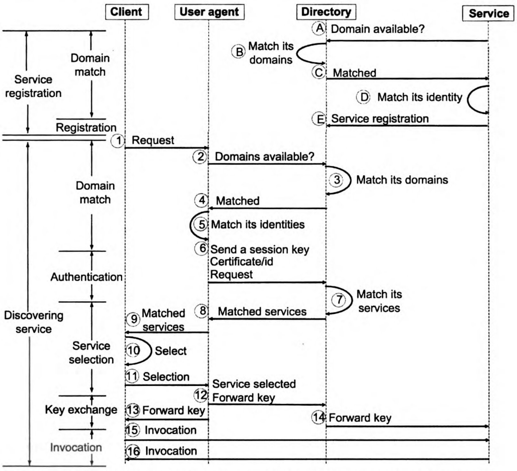

Figure 5-6. The PrudentExposure message sequence diagram.

5.4.7. Putting It All Together -- The Detailed Mechanism and Protocol

The message exchange sequence diagram of our model is shown in Figure 5-6 and the protocol is shown in Figure 5-7. The top halves in the figures are the service registration part, while the bottom halves are the discovering service part. In the first step of service registration (step A in Figure 5-6), a service utilizes the same approach as we discussed in Section 5.4.3, 5.4.4, and 4.5: the service generates a Bloom filter using the domain identity and mixes the true code word with randomly selected bits. (An owner may use two different domain identities for users and services, respectively.) If a directory finds a match in step B, it replies with another Bloom filter in step C. Note the directory sets two bits: one bit is that the directory finds itself as a match and the other bit is set by using another hash function as we have discussed in Section 5.4.2. When the service also finds a match, it sends a service registration message via the secure channel as we discussed in Section 5.3.1 (step D and E in Figure 5-6). Afterwards, the service periodically updates its service information. As long as updates succeed, the service does not send Bloom filters to discover directories. On the other hand, if the service fails to update service information or restarts, it may send out Bloom filters to discover directories.

In the first step of the discovering service part, a client sends a service request to a user agent using a session key shared between them as we described in Section 5.3.2. Then in step 2, the user agent sends a message asking for available domains. After receiving the request, a directory does a membership test to see if its domain matches the request in step 3. If the directory finds a match in the Bloom filter, it creates a new Bloom filter and then sends the filter back in step 4.

If there is a match in step 5, the user agent authenticates with the directory. All messages afterwards are encrypted using a session key exchanged between the user agent and the directory. The user agent forwards the service request in step 6. Then the directory matches the services and sends a reply message back in steps 7 and 8. Next, the user agent forwards the messages to the client and lets the client select (steps 9 and 10).

After the client selects a service, it sends the request to the user agent and the user agent forwards it with another session key to the directory (steps 11 and 12). This session key is for the client and the service to use. Next, the user agent and the directory forward the session key to the client and the service, respectively (steps 13 and 14). Last, in steps 15 and 16, the client and the service interact with each other.

Notation:
C is a client; U is a user agent; D is a directory; S is a service.
M is a message.
t_X is a timestamp that X attaches.
R_X is a random number that X generates.
SR is a service request. MS is a matched service list.
PS is a service the client picks.
K_{XY} is a symmetric encryption key shared between X and Y.
UBF is a Bloom filter representing the domains that a user belongs.
SBF is a Bloom filter representing the domain that a service belongs.
DBF is a Bloom filter representing the domain a directory in charge of.
$()_{KX}{}^{-1}$ is X's signature using its signing private key.
$()_{KX}$ is an encryption using the encryption public key of X.
$()_{KXY}$ is an encryption using a symmetric key K shared between X and Y.

Msg	Sndr/Rcvr	Message	Step in Fig 6
a	S→D:	SBF, R_S, t_S	A
b	D→S:	DBF, R_S, t_S, (DBF, t_S))$_{KD}{}^{-1}$	C
c	S→D:	(M, t_S)$_{KDS}$	E
1	C→U:	(SR, t_C)$_{KUC}$	1
2	U→D:	UBF, R_U, t_U	2
3	D→U:	DBF, R_U, t_U, (DBF, t_U))$_{KD}{}^{-1}$	4
4	U→D:	(K_{UD}, t_U)$_{KD}$, (U, t_U, K_{UD})$_{KU}{}^{-1}$, (SR, t_U)$_{KUD}$	6
5	D→U:	(MS, t_U, t_D)$_{KUD}$	8
6	U→C:	(MS, t_{U2}, t_C)$_{KUC}$	9
7	C→U:	(PS, t_{C2}, t_{U2})$_{KUC}$	11
8	U→D:	(PS, t_D, K_{CS})$_{KUD}$	12
9	D→S:	(K_{CS}, t_{D2})$_{KDS}$	13
10	U→C:	(K_{CS}, t_{C2})$_{KUC}$	14
11	C→S:	(M_1, t_{C3})$_{KCS}$	15
12	S→C:	(M_2, t_{C3})$_{KCS}$	16

Figure 5-7. The PrudentExposure protocol.

5.5. System Evaluation

In this section, we first analyze the mathematical properties of our PrudentExposure model. Next, we consider the possible threats. Then, we formally verify our security protocol. Last, we discuss performance issues.

5.5.1. Mathematical Properties of the PrudentExposure Model

Bloom filter membership tests always recognize members correctly [19]. Thus, our method is complete, such that matches in the original form are always matched in the Bloom filter form. However, the membership tests can have false positives, which mean non-members are recognized as members. When this happens, a waste of processing and communication occurs and privacy information may leak. If there is a false positive case at the directory side, a Bloom filter will be returned. If there are false positive cases at the user agent side, identities are supplied for authentication. In our case, the false positive rate is quite low. For example, if a filter is 8K bits long with 100 bits set, the false positive rate at the directory side is about 0.0124 given that the query is not sent by a domain user; and at the user agent side is less than 0.0001 given that the directory does not represent a domain that the user wants to discover. Increasing the length of Bloom filters, using more hash functions, or exchanging more rounds of Bloom filters can further decrease the false positive rate. More detailed discussion of the false positive rate in Bloom filters may be found in [72].

The mod function, *mod(8192)*, is a mapping function from 2^{128} space to 2^{13} space (MD5 is 128 bits, others are 160 bits). This means that 2^{115} possible hash results will set the same bit in a Bloom filter (supposing the hash result is evenly distributed in the 2^{128} space). It is not worthwhile for attackers to determine the hashes. Even if an attacker found a hash, it is still mathematically impossible to find the original domain identity from the hash.

5.5.2. Threats Analysis

Since user agents and directories have a priori knowledge, such as public/private keys or user names and passwords, they can set up session keys. Thus, all the messages they exchange can be encrypted, as shown in Figure 5-6 step 6 and afterwards. Hence, we focus on how a user agent correctly identifies directories and how a directory correctly identifies a user.

Unlike the discussion in Section 5.5.1 that assumes both directories and user agents are honest, user agents and directories might be malicious. There could be external or internal attackers. To prevent internal attackers, we may use the technique discussed in Section 5.4.4 and 5.4.5.

External attackers on the other hand will not gain much because the steps 2-5 in Figure 5-6 are for a user agent to identify the existence of the directories. Acting as a user agent, the attacker may cause a directory to respond. In this case, the probability of a false positive using one hash function for the directory is

$$p(falsepositive \mid nonmember) = \frac{M}{L},$$

where M is the number of bits set in a Bloom filter and L is the filter length. Since a large number of bits set causes a higher false positive rate, a very high ratio of 1's set in a Bloom filter is suspicious. Directories check the number of bits set in a filter. If the number of the bits set is more than a threshold (5%), directories do not further check whether there is a match in the filter. By default, the filter length is 8192 bits and 5%

allows a user to set up to 410 code words. Similarly, an attacker that keeps sending Bloom filters with a majority of the bits set differently is also suspicious. Directories do not remember the states before authentication. Therefore the attacks will cause a limited waste of resources.

Acting as a directory, the chance of guessing the correct answer is

$$p(falsepositive \mid nonmember) = \frac{1}{L}.$$

Moreover, if the signature is required, as we discussed in Section 5.4.4, it is considered computationally difficult for an attacker to sign a reply message. Another possible attack is to replay reply messages heard from other directories. A replay message, however, is easy to detect, since a user agent has already seen it from the genuine directory.

5.5.3. Formal Verification

We formally verify our protocol using BAN logic [65] and extend the logic to meet our needs. It assists us to improve our protocol. During a few rounds of the design and verification processes, the logic helped us find a subtle bug. The detailed notation may be found in [65]. As a convention of the verification process, we first convert the protocol to an idealized protocol. Next, we list all the assumptions. Then, we deduct step-by-step based on the logical postulates to reach our conclusion. The deduction is quite lengthy. We show the idealized protocol and stepwise results in Table 5-1. (The verification of the service registration part is very similar to the step 2, 3, and 9 of the discovering service part, thus we omit it.) Step 1 is trivial. Since Step 4 to Step 11 is a procedure of authentication and key distribution, the use of BAN logic is straightforward.

We show the most complicated step, step 4, in Table 5-2, and omit the similar and lengthy discussion of the other steps.

Table 5-1. Formal verification using BAN logic.

Step	Idealized Protocol	Stepwise results
1	$\{SR, \#(SR)\}_{KUC}$	$U \models SR$
2	UBF	$D \models UBF$
3	DBF, $\{DBF, t_U\}_{KD}{}^{-1}$	$U \models DBF$
4	$\{U \xleftrightarrow{K_{UD}} D, \#(U \xleftrightarrow{K_{UD}} D)\}_{KD}, \{t_U, U, U \xleftrightarrow{K_{UD}} D\}_{KU}{}^{-1}, \{SR, \#(SR)\}_{KDU}$	$D \models U \xleftrightarrow{K_{UD}} D$, $D \models SR$
5	$\{MS, \#(MS)\}_{KUD}$	$U \models MS$
6	$\{MS, \#(MS)\}_{KUC}$	$C \models MS$
7	$\{PS, \#(PS)\}_{KUC}$	$U \models PS$
8	$\{PS, \#(PS), C \xleftrightarrow{K_{CS}} S, \#(C \xleftrightarrow{K_{CS}} S)\}_{KUD}$	$D \models PS, D \models C \xleftrightarrow{K_{CS}} S$
9	$\{C \xleftrightarrow{K_{CS}} S, \#(C \xleftrightarrow{K_{CS}} S)\}_{KDS}$	$S \models C \xleftrightarrow{K_{CS}} S$
10	$\{C \xleftrightarrow{K_{CS}} S, \#(C \xleftrightarrow{K_{CS}} S)\}_{KUC}$	$C \models C \xleftrightarrow{K_{CS}} S$
11	$\{C \xleftrightarrow{K_{CS}} S, \#(C \xleftrightarrow{K_{CS}} S)\}_{KCS}$ from C	$S \models C \models C \xleftrightarrow{K_{CS}} S$
12	$\{C \xleftrightarrow{K_{CS}} S, \#(C \xleftrightarrow{K_{CS}} S)\}_{KCCS}$ from S	$C \models S \models C \xleftrightarrow{K_{CS}} S$

Since steps 2 and 3 are not authentication messages, the logic cannot be used directly. We extend the logic to help us check these two steps because the power of the BAN logic is its ability to check the freshness and binding. We extend the logic constructs as follows:

- **($M \subset G$):** M is a member of group G, who knows the shared secret.

- **P[BF]Y:** P finds a match in a Bloom filter, which uses Y as a secret. This only means that there is a possibility that the Bloom filter generating party knows the secret Y. The probability is as we discussed in Section 5.5.1 and Section 5.5.2.
- **$P \overset{Y}{\infty} G$:** P shares a secret Y with a group G. In our case, G is the group of users of the domain P.

We also add the following postulates:

$$\frac{P \overset{Y}{\infty} G,\ P[BF]_Y}{P \models (M \subset G) \mid\sim BF} \qquad (1)$$

$$\frac{P \models (M \subset G) \mid\sim BF,\ \#(BF)}{P \models (M \subset G) \models BF} \qquad (2)$$

$$\frac{P \models (M \subset G) \models BF,\ (M \subset G) \mid\Rightarrow BF}{P \models BF} \qquad (3)$$

The first postulate, (1), states that if P finds a match in a Bloom filter using secret Y, then there is a probability that one member M of group G generates the Bloom filter. Therefore, we are clear that if the Bloom filter is generated without a time variant parameter, the filter may be replayed. The second postulate, (2), goes further based on the freshness of the Bloom filter. It comes to the conclusion that P believes that one member of its user group generates the filter with a certain probability. Since the member has the control over the generation of the Bloom filter, P believes the Bloom filter (Postulate (3)). Based on these postulates, we can mechanically deduct and get the results for step 2 and 3 as shown in Table 5-1.

Table 5-2. Verification of step 4 in Table 5-1. The micro steps 4 and 7 in this table are shown in Table 5-1 as stepwise results.

Micro steps	Deduction	Notes
1	$\dfrac{D \lhd \{K_{UD}\}_{K_D}, D \models \xrightarrow{K_D} D}{D \lhd K_{UD}}$	Using the other rule. The directory sees the session key that the user agent wants to share with it.
2	$\dfrac{D \lhd \{K_{UD}\}_{K^{-1}}, D \models \xrightarrow{K_U} U}{D \models U \mid\sim K_{UD}}$	Using the message-meaning rule. The directory believes that the user agent once said the session key.
3	$\dfrac{D \models U \mid\sim K_{UD}, D \models \#(K_{UD})}{D \models U \models K_{UD}}$	Using the nonce-verification rule. The directory believes that the session key is recent and the user agent still believes in it.
4	$\dfrac{D \models U \models K_{UD}, D \models U \mid\Rightarrow K_{UD}}{D \models K_{UD}}$	Using the jurisdiction rule. The directory believes the session key.
5	$\dfrac{D \models U \overset{K_{UD}}{\leftrightarrow} D, D \lhd \{SR\}_{K_{UD}}}{D \models U \mid\sim SR}$	Using the message-meaning rule. The directory believes the user agent once said the service request.
6	$\dfrac{D \models U \mid\sim SR, D \models \#(SR)}{D \models U \models SR}$	Using the nonce-verification rule. The directory believes that the service request is recent and the user agent still believes in it.
7	$\dfrac{D \models U \models SR, D \models U \mid\Rightarrow SR}{D \models SR}$	Using the jurisdiction rule. The directory believes the service request.

Moreover, the logic forces us to explicitly write down our assumptions to clarify our design goals. Our protocol is targeted for wired or wireless LAN environments. If used beyond LAN environments, given that a user agent and a directory cannot directly hear each other's broadcast or multicast messages, the messages may be replayed in real-time without notice. Furthermore, the time stamp and the random number used as the time variant parameter require accurate internal clocks in the user agents and directories. (The clocks do not drift hours away. Otherwise, user agents and directories have to use large caches to check the validity of the timestamps.)

5.5.4. Performance Discussion

Our model is based on the client-service-directory model. Compared to that model, our model is more efficient because a directory only replies when a match is found. However, three additional messages are needed in comparison to the insecure client-service-directory model. Two messages are for the user agent and the directory to send the session key to the client and the service, respectively. The other message is for the user agent to notify the directory the selected service and the session key. The three messages should not introduce much overhead.

Our concerns are the overheads that user agents and directories need to calculate Bloom filters and do membership tests (steps 2-5 in Figure 5-6); and that every party does symmetric key encryption and decryption operations (from step 6 to step 16).

We measure our prototype system on Compaq iPAQs running Microsoft PocketPC 3.0. Each PDA has an ARM SA1110 206 MHz processor, 64MB RAM, an expansion pack, and a D-Link DCF-650W wireless card. The wireless cards are set to the 802.11 ad hoc mode with 2Mbps. Our software is developed using Microsoft eMbedded Visual C++ 3.0.

Table 5-3. Performance evaluations on Bloom filters exchanged between user agents and directories.

Component	Task	Time	Step in Figure 5-6
User agent	Generate a Bloom filter with 100 dynamic code words	15.62ms	2
User agent	Send a Bloom filter	2.52ms	2
Directory	Check whether there is a match in the Bloom filter	<1ms	3
Directory	Find a match and generate and send a Bloom filter	5.06ms	4
User agent	Waiting time after sending a filter until receiving a reply	11.62ms	
User agent	Check whether there is a match in the Bloom filter	<1ms	5

We measure the performance of user agents and directories exchanging Bloom filters (steps 2-5 in Figure 5-6). In our prototype, this part of the protocol uses UDP multicast. (Multicast or broadcast communication may not be reliable. More complex algorithms may be used to improve reliability such as the Multicast Convergence Algorithm in SLP [11].) User agents and directories are configured to use a multicast address for query messages and another multicast address for reply messages. Table 5-3 shows the average time of the tasks in 100 discovery processes. It takes a user agent about 20 milliseconds to generate a Bloom filter with 100 code words and to send a query message. A directory takes less than 1 millisecond to check whether there is a bit that matches its code word. If there is a match, a directory spends about 5 milliseconds to generate another Bloom filter and send back a reply message. Next, it takes less than 1 millisecond for a user agent to check whether the Bloom filter in the reply message matches the code word. In summary, it takes a user agent about 30 milliseconds from the time to generate a filter to finish processing the first reply message.

From step 6 to step 16 in Figure 5-6, when compared to an insecure model, every step needs some symmetric key encryption and decryption operations. Our previous experience with respect to building a secure service discovery protocol shows that the overhead to do secure key operations are rather efficient on PDAs [68]. The public key operations take hundreds of milliseconds, while symmetric key operations and hash operations take a few to dozens of milliseconds. Thus, we believe that discovering a service should be possible within a reasonable time period.

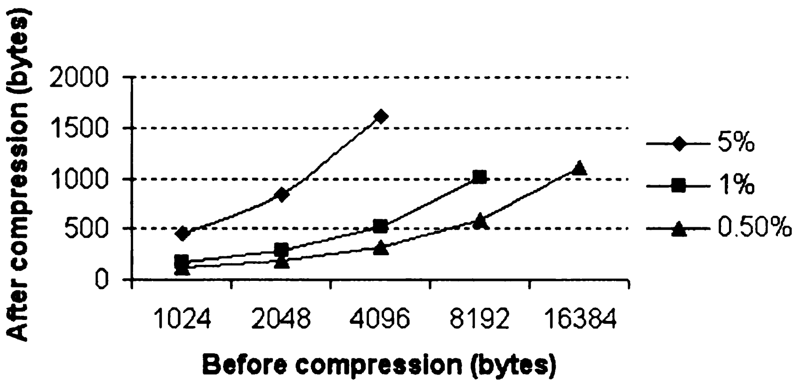

Figure 5-8. Simulation results of the compression ratio of Bloom filters with different percentage of bits set.

The compression ratio of the Bloom filters that we use is high, as shown in Figure 5-8. We randomly generate 1K, 2K, 4K, 8K, and 16K bytes Bloom filters with 5%, 1%, and 0.5% of the bits set in the filters. We generate 1000 filters for each combination of the rate and length. Next, we use zlib (a data compression library) version 1.1.4 for Windows CE and select the maximum compression option to compress the filters. Last, we determine the largest file size for each combination of the rate and length. The compression takes from 20 milliseconds for 1K bytes filters up to 600 milliseconds for

16K bytes filters on the same PDAs as we described above. Thus, we may compress Bloom filters in the protocol without affecting the performance much.

5.6. Summary

In this chapter, we have proposed a private, secure, and efficient service discovery model. The PrudentExposure model provides an efficient way for authorized users to discover services, while hiding services from unauthorized users. It automatically selects the right identities for service accesses. We protect sensitive information such as service information, user identities, user's presence information, domain identities, owner's presence information, and service requests. We have analyzed our model mathematically and have verified our protocol formally. Our prototype system shows that our method is efficient.

To further enhance usability, our model needs mechanisms to monitor and control the accesses of the services and thus provide a method for users to easily view the status of the services and the history of service accesses. To coexist with the current models (providing services without a privacy concern), the security checks at the directories and services may be loosened. One possible solution is to reserve bits in the Bloom filters to represent domains that do not need identities for accesses. Meanwhile, the directories will not check user privileges and always reply with the availability of the services.

Revocation of a domain identity from a user is a problem, when many users share the domain identity as a secret. This issue is addressed in the next chapter.

CHAPTER 6. PRIVATE AND SECURE SERVICE DISCOVERY VIA PROGRESSIVE AND PROBABILISTIC EXPOSURE

6.1. Introduction

In Chapter 5, we proposed PrudentExposure service discovery approach, such that only users and service providers that share secrets discover and communicate each other. However, there is still privacy leak among insiders. For example, if Bob just shares an MP3 player with Alice, it is unnecessary to contact Bob when Alice discovers an electronic book. In this chapter, we improve personal privacy based on the PrudentExposure model and only expose sensitive when the exposure is necessary.

After further analysis of privacy concerns, we classify them into four cases as shown in Table 6-1. The four cases are the combinations resulting whether a user and a service provider have privacy concerns. Since there is no privacy concern in case 1, we may directly apply authentication and authorization to secure services. In case 2, service providers may announce their service information first because they do not have privacy concerns. Note that service providers can announce service information in an encrypted form, so only users who can decrypt messages will understand service information. If a service provider does not provide a desired service, a user may keep silent and therefore protect the user's privacy. Similarly, in case 3, users send their requests first. If a service provider provides the required service and the user has privilege, then the service provider contacts the user. Otherwise, the service provider keeps silent. Nevertheless, we identify that case 4 is as difficult as a chicken-and-egg problem. That is, neither users nor service providers want to expose their information first when both parties have privacy concerns. Thus, users and service providers may not even want to communicate with each other. To the best of our knowledge, this is a new problem, and we have not yet seen any solution.

Table 6-1. Four cases of privacy concerns.

Service provider / User	No	Yes
No	Case 1	Case 3
Yes	Case 2	Case 4

In this chapter, we propose a progressive approach to solve the chicken-and-egg problem. Users and service providers exchange partial and encrypted forms of their identities and service information. They establish mutual trust over multiple rounds of communication. Any mismatch during the procedure stops the communication and indicates that further interaction is unnecessary. Because only partial information is exposed in the unnecessary cases, sensitive information is not exposed in an understandable way to inappropriate participants. Via simple strategies, users and service providers know the number of communication rounds and the number of bits to exchange in each round to reach mutual trust.

Similar to PrudentExposure, in this paper, a user uses a program to manage credentials. The program supplies necessary credentials for a user. Unlike PrudentExposure, the progressive approach uses a different method to exchange code words and service information. More importantly, the new approach avoids unnecessary privacy exposure. Figure 6-1 illustrates the different design goals of PrudentExposure and the progressive approach from a user's point of view. From a service' providers point of view, the diagram is similar. PrudentExposure discovers users and service providers that share secrets. The progressive approach discovers a smaller set: users and service providers that share secrets, provider services, and have privileges to access services.

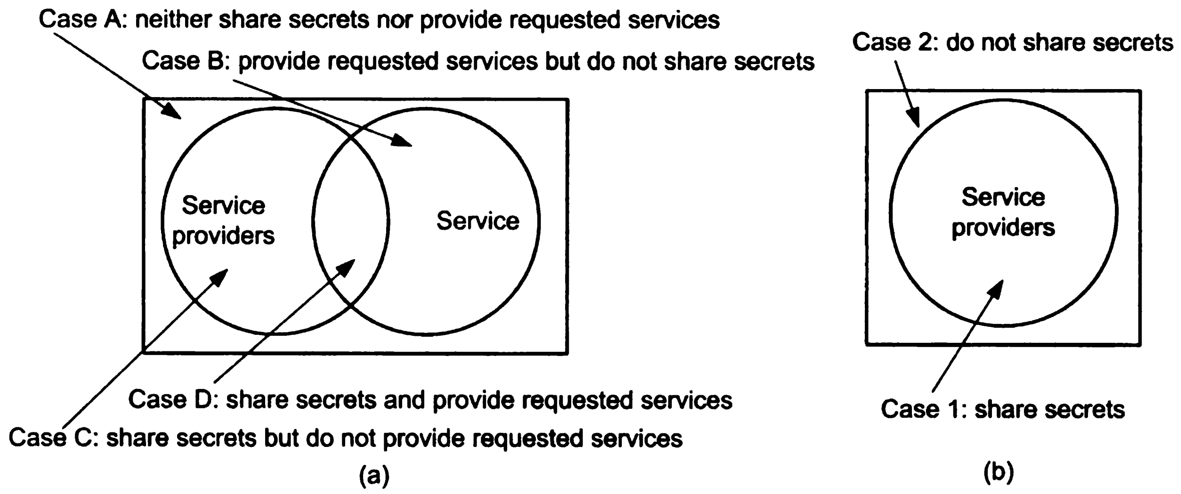
Case A: neither share secrets nor provide requested services
Case B: provide requested services but do not share secrets
Service providers
Service
Case D: share secrets and provide requested services
Case C: share secrets but do not provide requested services
(a)
Case 2: do not share secrets
Service providers
Case 1: share secrets
(b)

Figure 6-1. Different design goals of PrudentExposure and the progressive approach from a user's point of view. (a) The design goal of the progressive approach is to find Case D. (b) The design goal of PrudentExposure is to find Case 1.

6.2. A Progressive Exposure Approach

In this chapter, we use *user* to represent the discovering party, which includes a client device and a user or a program that supplies the user's identities for the user; whereas a *service provider* includes services owned by a person or an organization and computers that service owners use to conduct authentication and authorization.

To solve the chicken-and-egg problem, users and service providers expose partial (several bits of) encrypted information in turn as shown in Figure 6-2. Users partially provide their identities and service requests, and service providers partially provide their identities and service information. During each round of message exchange, both the user and service provider verify the partial information. If there is a mismatch, the communication stops. If matches repeatedly occur, at some point, the service provider and the user believe with high probability that the other party is legitimate. Finally, the two parties establish a connection for service usage. In this chapter, we define that a user is legitimate if the user shares a secret with the service provider and the user has privilege to access the service. Similarly, a service provider is legitimate if the service provider shares a secret with a user and provides the requested service type.

During each round, the progressive protocol identifies and excludes unnecessary exposures. For the unnecessary exposure cases, sensitive information is preserved because only a few bits of sensitive information are exposed. For instance, a service provider may provide a small set of service types in comparison to all possible service types. Suppose a unique ID represents each service type and all IDs have the same length. On average, each bit of a service type ID excludes half of the service types provided by a service provider. Thus, an unnecessary exposure is identified in several bits of service information exchange. However, to learn the requested service type and available service types, the whole IDs are needed.

Moreover, when a mismatch is found in a message, neither a user nor a service provider is certain about the true reason of the mismatch. That is users and service providers cannot discern Case A, Case B, or Case C in Figure 6-1. For example, a user finds a match on identity information and a mismatch on service information, but maybe the true reason is a false positive match on the identity information and a mismatch on the service information. So, in the example, the mismatch looks like Case B, but it is Case A.

In the remainder of this section, we first discuss how to exchange encrypted sensitive information, so users and service providers that do not share secrets do not even acquire partial sensitive information. Then, we present our security protocol. Next, we illustrate that unnecessary exposure can be quickly detected by exchanging a few bits. An analysis of the unnecessary exposure in terms of the probabilities is given. The probabilities are known for each state in the communication. Last, we show the strategies that users and service providers use to expose their information.

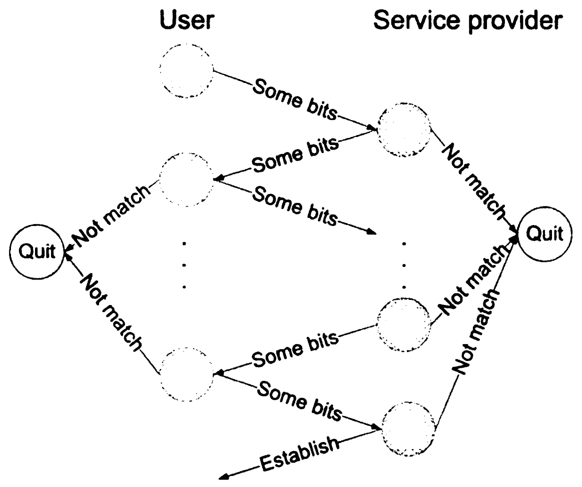

Figure 6-2. Users and service providers expose partial identity and service information in turn.

6.2.1. Exchange Identity Information in the Code Word Form

Users and service providers exchange code words without explicitly specifying their identities in the messages. A code word is generated from a unique secret shared between a user and a service provider. A user says several bits of a code word and a service provider checks. If the service provider does not recognize the code word, he keeps silent. Otherwise, he says another several bits of the code word and the user checks. So, eavesdroppers do not understand who is talking to whom.

To protect against replay attacks, a user and a service provider speak one-time code words. During the code word generation, we use a time variant parameter (TVP), which consists of a time stamp and a random number. A TVP is transmitted from a user to a service provider in the first message. Thus, each time a user and a service provider speak a different code word and a replay attack can be easily detected. (Our approach requires loosely synchronized clocks.)

The left side of Figure 6-3 illustrates the generation of a code word. A TVP and a unique secret shared between a user and a service provider are the two inputs to a hash function. Specifically, we use hash-based message authentication codes (HMAC) proposed in [71]:

h(Secret, XOR padding1, h(Secret, XOR padding2, Time Variant Parameter)).

MD5 is used in place of *h()*. Therefore, a code word is the hash result. (The right side of Figure 6-3 is discussed in the next section.)

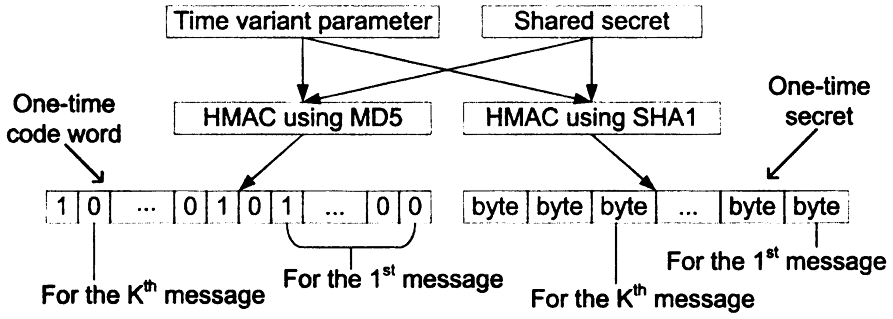

Figure 6-3. Generating a one-time code word to identify a user and a service provider (left side); and generating a one-time secret to protect service information (right side).

The nature of the HMAC ensures that it is computationally difficult to find the shared secret from the hash results [58]. Thus, only a user and a service provider who share a secret can correctly generate and verify the code words.

6.2.2. Exchange Encrypted Service Requests and Available Service Types

Besides code words, users and service providers also exchange several bits of service information in the messages. Users say their service requests and service providers say the available service types. Only if the bits of a code word and service information match, one provides more bits.

Services are identified by unique IDs with the same length. Users and service providers use one-time secrets to protect service information. The right side of Figure 6-3 shows the generation of a one-time secret using SHA-1 in place of *h()*. Unlike a code word that uses bits of the hash result, in each message, a user and a service provider use a byte to encrypt and decrypt the service information. To be precise, the encryption is $cipher = service \oplus one\text{-}time\ secret$, and the decryption is $service = cipher \oplus one\text{-}time\ secret$. The encryption method is known as the Vernam cipher [58]. According to [58], if the bytes (generated one-time secrets) that we use to encrypt service information are random, our encoding method is computationally secure. We show the bytes are random and follow the uniform distribution over an integer set in Section 6.3.4.

Table 6-2. The encoding scheme for service types.

Next one bit	Coding
0	00
1	01
0 and 1	10

(a)

Next two bits	Coding
00	0000
01	0001
10	0010
11	0011
00 and 01	0100
00 and 10	0101
00 and 11	0110
01 and 10	0111
01 and 11	1000
10 and 11	1001
00, 01, and 10	1010
00, 01, and 11	1011
00, 10, and 11	1100
01, 10, and 11	1101
00, 01, 10, and 11	1110

(b)

At a service provider's side, since only partial information is exchanged, more than one service types with the same initial bits may match a user's request. For example, a user requests a service type: 10001; a service provider has two service types: 10001 and 10101. If a user says the first two bits, 10, the service provider will find two service types start with 10. To inform the user that there is more than one service type that matches his request, we encode the possible combinations of one and two bits as shown in Table 6-2 (a) and (b), respectively. In our example, if the service provider replies with one bit of service information, he will send the coding 10. If the service provider replies with two bits of service information, he will send the coding 0101.

6.2.3. The Security Protocol

A user starts a discovery process. Without knowledge of the existing services and service providers, he may specify all code words and the encrypted service request along with a TVP. Figure 6-4 (a) shows the message format. A user may include up to three hundreds of pairs of code words and service information in one network packet. In the following messages, a user and service provider exchange 1 or 2 more bits of the code word and service information as shown in Figure 6-4 (b). (The number of bits in the messages will be discussed in detail in Section 6.2.4)

The first message is sent as a broadcast message or a UDP multicast message for minimum configuration overhead. The following messages between a user and a service provider are sent via TCP unicast to guarantee delivery. In addition, a service provider indicates the code word for which he finds the match in the second message.

The discussion so far is based on the condition that a user and a service provider share a unique secret. When a user interacts with many service providers and a service provider has many users, false positive matches are very likely to happen because only partial information is exchanged initially. The problem might be addressed for a service provider by sharing a secret among all users. The shared secret, however, is difficult to revoke from an individual user. Our solution is that service providers and users may use two types of shared secrets: domain secrets, and user secrets. A domain secret is used in the first message to identify a service provider. A user secret is used in the following messages to identify a user within a service provider. Figure 6-4 (c) shows that a service provider specifies a list of code words generated from user secrets in the second message along with a TVP that he selects. Afterward, the user indicates the matched code word and the two parties exchange bits of information as shown in Figure 6-4 (b). To revoke a user's service discovery privilege, the user secret is invalidated by the service provider, while updating the domain secret may not be imminent.

Figure 6-5 shows the protocol that uses a domain secret and a user secret. After the first three messages, a user and a service provider send messages in the same format as shown in message 4. The process continues until either a mismatch is found or legitimacy reaches a high probability. If a mismatch is found, a message indicating that the communication stops is sent to the other party. If high legitimacy is found, the service provider instructs the user to prepare for service access.

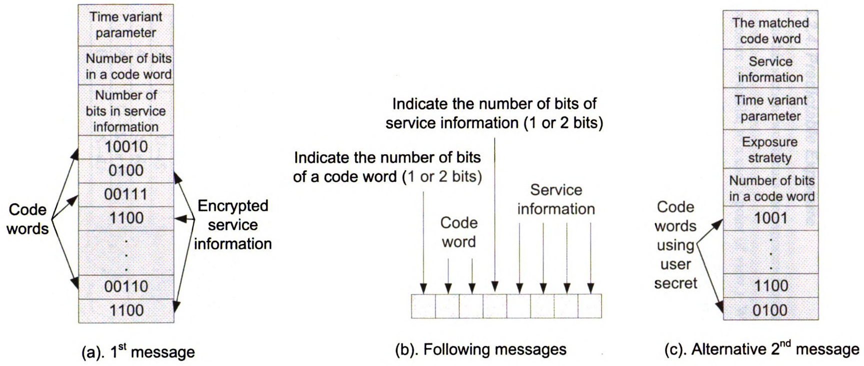
Time variant parameter
Number of bits in a code word
Number of bits in service information
10010
0100
00111
1100
00110
1100
Code words
Encrypted service information
(a). 1st message
Indicate the number of bits of service information (1 or 2 bits)
Indicate the number of bits of a code word (1 or 2 bits)
Code word
Service information
(b). Following messages
The matched code word
Service information
Time variant parameter
Exposure stratety
Number of bits in a code word
1001
1100
0100
Code words using user secret
(c). Alternative 2nd message

Figure 6-4. Message formats.

6.2.4. Predictable Exposure

In this section, we discuss the exposure from a service provider's point of view. (The exposure is similar to a user's point of view.) When verifying code word bits, a service provider finds either a match or a mismatch. If a mismatch occurs, he knows the other party does not know the shared secret (not user). If a match happens, he does not know

Notation:

U is a user; S is a service provider.

t_X is a timestamp that X attaches. R_X is a random number that X attaches. (X is a user or a service provider.)

CB_{SUI} is bits of a code word generated from the domain secret shared between U and the ith S.

CW^J_{SUL} is bits of a code word generated from the user secret shared between Lth U and S in the Jth message.

KUSI is a secret generated from the domain secret that U and Ith S use to encrypt and decrypt messages.

KUSL is a secret generated from the user secret that Lth U and S use to encrypt and decrypt messages.

SRB^J is bits of the requested service information in the Jth message.

SAB^J is bits of the available service information in the Jth message.

ST is an exposure strategy.

K_{USI} is a symmetric encryption key generated at U and the ith S.

$\{\}^N$ is a set of N elements. (N is an integer that is greater than zero.)

Q is a message indicates the communication stops.

P is a message instructs a user to prepare for service usage.

Message	Sender/Receiver	Message
1	U→S:	$R_U, t_U, \{CB^1_{SUI}, (SRB^1)_{KUSI}\}^N$
2	S→U:	$R_U, t_U, R_S, t_S, CB^1_{SUI}, (SAB^2)_{KUSI}, \{CW^2_{SUL}\}^M, ST$
3	U→S:	$R_S, t_S, CW^2_{SUL}, CW^3_{SUL}, (SRB^3)_{KUSL}$
4	S→U:	$R_S, t_S, CW^4_{SUL}, (SAB^4)_{KUSL}$
A	U→S: or S→U:	R_S, t_S, Q
B	S→U:	R_S, t_S, P

Figure 6-5. The security protocol using domain secrets and user secrets.

whether the match is a false positive match because only part of a code word is compared. Therefore, given a match is found in a message, a service provider is interested in the probability $p(not\ user \mid match)$, that is, what is the probability that the other party does not know the shared secret? It depends on two probabilities: the probability of false positive matches, $p(match \mid not\ user)$, and the probability that a message comes from a user who knows the shared secret, $p(user)$.

The first probability, $p(match \mid not\ user)$, depends on our design: the number of code words a user has, and the number of bits exposed so far. Assuming that the last several bits of the code words follow the uniform distribution over an integer set, the probability in the first message is:

$$p(match \mid not\ user) = 1 - \left(1 - \frac{1}{2^{number\ of\ bits}}\right)^{number\ of\ codewords} \qquad (1)$$

Given that a message is from one who does not know the shared secret, we want to control the false positive match rates. Thus, we may set the limit to 25%. A user may simply select the number of bits to expose from Table 6-3 based on the number of credentials that he has. A service provider examines the number of code words in the first message and the number of bits in a code word to learn the initial false positive rate. Afterwards, the false positive rate will decrease by half for each additional bit exchanged.

Table 6-3. Number of bits to expose in a code word vs. the number of credentials a user has.

Number of credentials	<5	<10	<19	<37	<74	<148	<295
Number of bits	4	5	6	7	8	9	10

The second probability, $p(user)$, is service provider dependent. It might be related to the environment and the mobility of a service provider. Based on history information, a service provider learns the probability that a discovery message is from his users, that is

$$p(user) = \frac{p(match) - p(match \mid not\ user)}{1 - p(match \mid not\ user)}$$

where $p(match)$ is the rate that the service provider finds a match in the first message; and $p(match \mid not\ user)$ in the first message is approximately 25% as we discussed above. Before a service provider accumulates enough history information, he may use a conservative strategy, for example, at least exchange 5 rounds of messages.

From the preceding two probabilities, we have

$$p(not\ user \mid match) = \frac{p(match \mid not\ user) \times (1 - p(user))}{p(match \mid not\ user)(1 - p(user)) + p(user)}.$$

Figure 6-6 shows that as the number of bits exchanged increases, $p(not\ user \mid match)$ decreases quickly. Moreover, a service provider with certain $p(user)$ knows the probability of $p(not\ user \mid match)$ in each round based on the number of bits exchanged.

Similarly, we calculate $p(not\ service \mid match)$. (*Match* means that a service provider or a user finds matches on the partial service information; and *not service* means that either a service provider does not provide the service type or a user does not have privilege.) It depends on three facts: the probability that the service provider has the service type $- p(service)$, the probability of false positive matches – $p(match \mid not\ service)$, and the number of service types that the service provider has. A user may not know how many service types are provided by a service provider, since available service types may change from time to time. So, the user by default sends four bits of the service request in the first message. The probability $p(not\ service \mid match)$ is calculated at the service provider's side. It is similar to the calculation of $p(not\ user \mid match)$. In addition, a service provider learns the $p(service)$ from history information.

We graph the relation between $p(not\ service \mid match)$ and number of bits exchanged after the first message for a service provider with 320 service types in Figure 6-7. If a service provider has 160 (or 640) service types, he needs to exchange one less (or more) message to reach the same probability.

Therefore, in each round a service provider knows the probability that a user is legitimate. Based on these probabilities, we design our exposure strategies.

6.2.5. The Exposure Strategies

Initially, a service provider chooses critical values for $p(not\ user \mid match)$ and $p(not\ service \mid match)$, for example 5% for both probabilities. If the probabilities are less than the critical values during a discovery process, the service provider considers that the user's legitimacy is high enough and instructs the user to prepare for service access.

A service provider does not need to calculate the probabilities to determine whether the legitimacy of a user reaches a high probability. Instead he only needs to perform table lookups.

Table 6-4 (a) and (b) list the number of bits required to reach the critical values for $p(not\ user \mid match)$ and $p(not\ service \mid match)$, respectively. For example, if a service provider has 80 service types, $p(user)$ is 0.016, and $p(service)$ is 0.032, then he needs to exchange 10 bits of the code word and 12 bits of service information. The numbers of bits in the tables are derived from the calculation results of the probabilities as we discussed in Section 6.2.4.

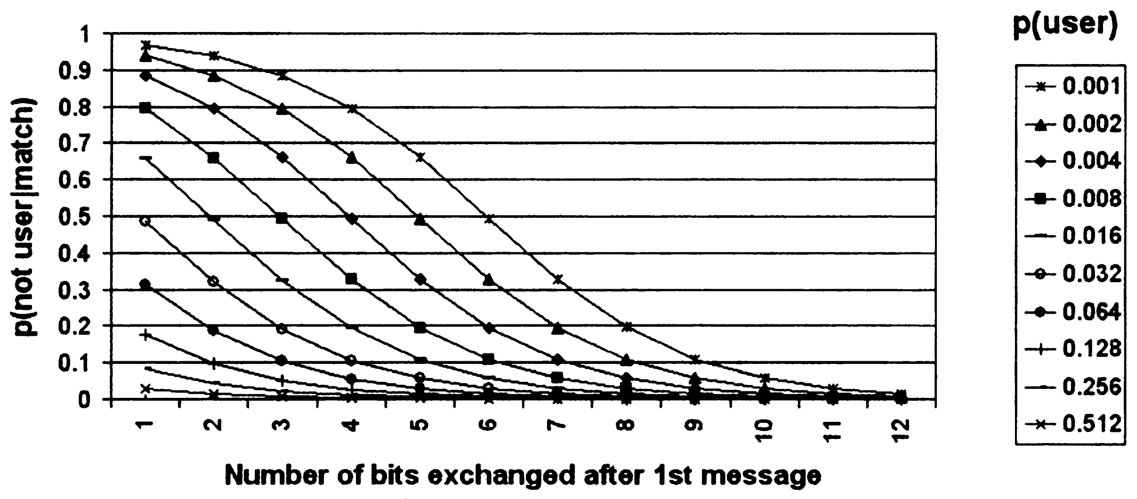

Figure 6-6. $p(not\ user \mid match)$ decreases as the number of code word bits exposure increase after the 1st message.

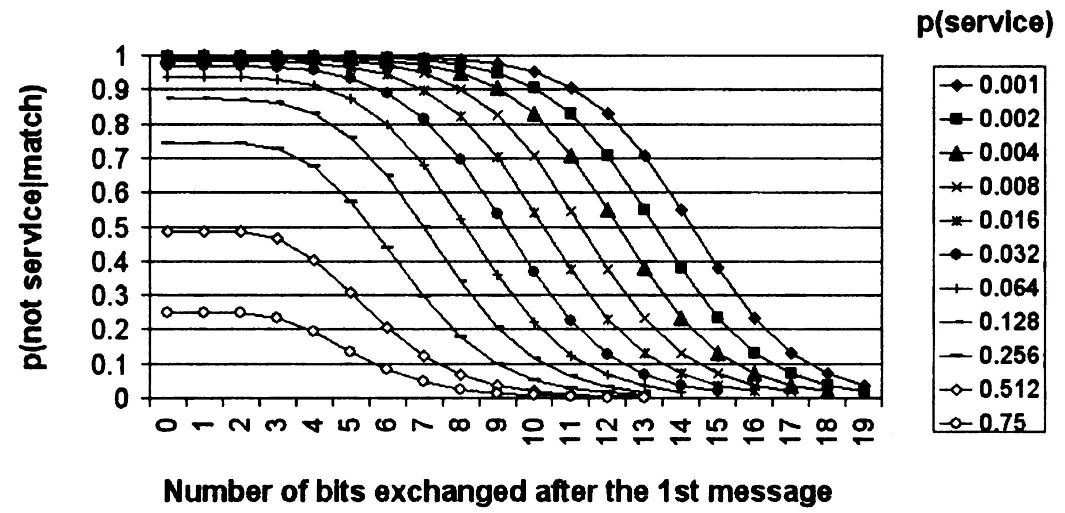

Figure 6-7. $p(not\ service \mid match)$ **decreases as the number of service information bits exchanged increases after the 1st message. (The graph shows a service provider with 320 services.)**

The general exposure strategy is that a service provider and a user exchange 1 or 2 bits of a code word and 1 or 2 bits of service information in one message, specifically three combinations: 1/1, 1/2, and 2/1. In each combination, the first number is the number of code word bits and the second number is the number of service information bits. Exposing 1 or 2 bits at a time is for the following two reasons. First, when mismatches occur, exchanging a few bits exposes minimal sensitive information. Second, a service provider may use different combinations to synchronize the convergence for $p(not\ user \mid match)$ and $p(not\ service \mid match)$ to reach the critical values at the same time. The disadvantage of the progressive exposure process is that the number of messages required to reach critical values may be large. However, our experiments show one message only takes about 4 milliseconds on a PDA. Thus, the communication overhead may not be a concern.

A service provider decides an exposure strategy for each discovery session based on the two numbers of bits. A user's strategy is to expose the same number of bits of the code word and service information as a service provider does. For example, if a service provider needs to exchange 10 bits of a code word and 12 bits of service information with a user, he may use the strategy 1/2, 1/1, 1/1, 1/1 and 1/1. After receiving a message, the user exposes the same number of bits that a service provider exposes. Thus, the interaction between a user and a service provider is 1/2, 1/2, 1/1, 1/1, 1/1, 1/1, 1/1, 1/1, 1/1, and 1/1.

Table 6-4. Number of bits to exchange to reach a critical value (less than 5%) for $p(not\ user \mid match)$ and $p(not\ service \mid match)$ in (a) and (b), respectively.

P(user)	0.001	0.002	0.004	0.008	0.016	0.032	0.064	0.128	0.256	0.512	0.75
No. of bits	14	13	12	11	10	9	8	7	5	4	2

(a)

P(service)	0.001	0.002	0.004	0.008	0.016	0.032	0.064	0.128	0.256	0.512	0.75
Service											
10	14	13	12	11	10	9	8	7	6	4	2
20	15	14	13	12	11	10	9	8	7	5	3
40	16	15	14	13	12	11	10	9	8	6	4
80	17	16	15	14	13	12	11	10	9	7	5
160	18	17	16	15	14	13	12	11	10	8	6
320	19	18	17	16	15	14	13	12	11	9	7
640	20	19	18	17	16	15	14	13	12	10	8
1280	21	20	19	18	17	16	15	14	13	11	9
2560	22	21	20	19	18	17	16	15	14	12	10

(b)

Up to now, the design of the strategies is from the service provider's perspective. From the user's perspective, he may trust a service provider's strategy for the following reasons. First, if a service provider does not know the shared secret, false positive matches occur and the service provider does not know the user's identity and does not understand the user's service request. The service provider wastes energy and processing power if he exchanges messages more than necessary. Second, if a service provider knows the shared secret and provides the requested service, exchanging messages more than necessary does not offer him any better payoff. Conversely, if the service provider exchanges messages less than necessary, he does not have enough confidence whether instructing a user for service access is necessary. Third, if a service provider knows the shared secret but he does not provide the requested service, he knows the user's identity and service request more accurately by exchanging messages more than necessary. However, the service provider also exposes his identity more precisely. The threat can be easily detected and is discussed in detail in Section 6.3.7.

6.3. System Evaluation

In this section, we first discuss three mathematical properties of our approach, namely the false positive overhead, code word conflicts, and convergence. Then, we present our experiments and hypotheses tests to verify that security properties hold. Next, we measure the performance of our protocol. After that, we formally verify binding and freshness of the protocol. Last, we analyze threats, attacks, and counter measurements.

6.3.1. False Positive Overhead Decreases Exponentially

To analyze the false positive overhead, we redraw the interaction between a user and a service provider (Figure 6-2) as a Markov chain illustrated in Figure 6-8. State *"0"* to *"k"* are transient states, and *"Quit"* and *"Establish"* are recurrent states. If both the bits of a code word and service information match, the process goes to the next state (exchange more bits). Otherwise, the process goes to the absorbing state, *"Quit"*. If both parties are legitimate, they always reach the "*Establish*" state. If at least one party is not legitimate (the exposure is unnecessary), the process should go to the *"Quit"* state. Since there are false positive matches, the process may go to the next state.

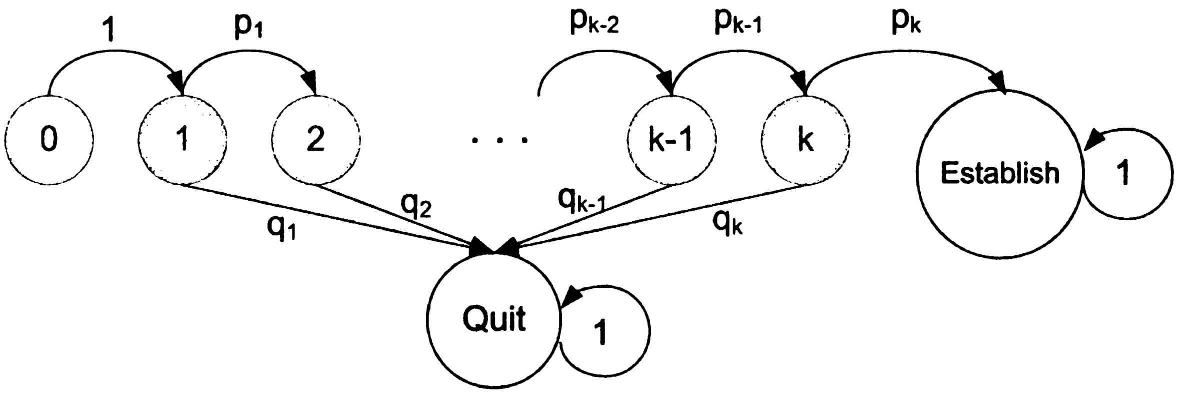

Figure 6-8. Message exchange process expressed as a Markov chain.

Suppose given an exposure is unnecessary, the probability of the process goes to the next state is p_i and the probability of the process goes to the *"Quit"* state is q_i. We calculate p_i from the following formula:

$$p_i = \frac{A+B}{C}, where$$
$$A = p(match \bigcap not\ user) \times p(match \bigcap not\ service)$$
$$B = p(match \bigcap user) \times p(match \bigcap not\ service)$$
$$C = p(not\ user) + p(user \bigcap not\ service)$$

Based on the calculation of the mean time spent in transient states [73], we calculate the probabilities that the Markov chain makes a transition into state "*i*" given it starts from state *"0"*:

$S = (I - P_T)^{-1}$

where S is a matrix of values of $s_{i,j}$, the time periods in state j given it starts in i.

$f_{0,j} = \frac{s_{0,j} - \delta_{0,j}}{s_{j,j}}$ *where $f_{0,j}$ is the probability in state j given it starts in "0".*

$$\delta_{0,0} = 1 \ \ and\ \delta_{0,j} = 0 \ when\ j \neq 0.$$

Therefore, the false positive rates are known for all states. The false positive rates decrease very quickly as more bits are exchanged. For example, if a service provider has 80 services, $p(user)$ is 0.016, and $p(service)$ is 0.032, and the strategy is 1/2, 1/2, 1/1, 1/1, 1/1, 1/1, 1/1, 1/1, 1/1 and 1/1. We calculate $f_{0,j}$. As shown in Table 6-5, false positive rates decrease exponentially.

Table 6-5. Probabilities that the process goes to the next state given an exposure is not necessary.

States	0	1	2	3	4	5	6	7
$f_{0,j}$	0	0.6143	0.2451	0.0347	0.0026	9.7E-05	1.9E-06	1.79E-08

6.3.2. Code Word Conflicts

In Section 6.2.4, we suggest that a user specify several bits of the code words in the first message. The code words for different service providers may have the same first several bits, called code word conflicts. The probability of a conflict is very high, but the expected numbers of conflicts are small. For example, if there are 50 code words and the first 8-bit of the code words are sent in the first message, there are about 4.5 code words have conflicts with other code words. Therefore, when generating code words, if a user finds the first several bits of a code word is already used for a service provider, he uses another TVP to generate code words again. In almost all cases, using two TVPs make the partial code words unique.

We prove that the expected value of code word conflicts is:

$$E(\textit{codeword conflicts}) = m\text{-}(1-(1-\frac{1}{n})^m)\times n,$$

where m is the number of code words and n is $2^{\textit{number of bits}}$.

PROOF. Let X_m be the number of distinctive partial code words (the first several bits of the code words in the first message) after we generate m code words. To find the expected number of code word conflicts, we have $E(\textit{codeword conflicts}) = m\text{-}E(X_m)$.

$$E(X_m) = E(X_m|Y = y_1)\times p_Y(y_1) + E(X_m|Y = y_2)\times p_Y(y_2)$$

where y_1 is the event that the first several bits of the m^{th} code word is the same as some of the code words that have generated, and y_2 is the event that the first several bits of the m^{th} code word are different from all code words that have been generated. So,

$$p_Y(y_1) = (\frac{E(X_{m-1})}{n}),\ p_Y(y_2) = (\frac{n-E(X_{m-1})}{n}), \text{and}$$
$$E(X_m|Y = y_1) = E(X_{m-1}),\ E(X_m|Y = y_2) = E(X_{m-1})+1.$$

Therefore, $E(X_m) = E(X_{m-1})\times(\frac{E(X_{m-1})}{n}) + (E(X_{m-1})+1)\times(\frac{n-E(X_{m-1})}{n})$

$$= (\frac{n-1}{n})\times E(X_{m-1})+1.$$

Since $E(X_1) = 1$, we have $E(X_m) = (1-(1-\frac{1}{n})^m)\times n$, after solving the recursive equation.

Thus, $E(\textit{codeword conflicts}) = m\text{-}(1-(1-\frac{1}{n})^m)\times n.$ ■

Similarly, the expected code word conflicts for different users of a service provider can be expressed in a similar formula and handled in the same way.

6.3.3. The Progressive Approach Converges

We prove the most complex case that many service providers coexist in a place; each service provider has many users; and a user and a service provider exchange 1 or 2 bits of information in each message.

PROPOSITION 1. *The discovery process converges.*

PROOF. First, we consider a discovery process between a user and one service provider, called a session. Since a service provider generates unique code words from user secrets (Section 6.3.2), a user and a service provider may establish up to one session. Each message exchanged in a session may be considered as a state in a Markov chain as shown in Figure 6-8. If there is a mismatch, one party quits. If both the code word and the service information match, one more message is exchanged. Each state transition causes the false positive rate to decrease because more bits of information are exchanged. After a finite number of state transitions, the probability is high enough so that only legitimate parties can generate the code words and then a service access is established (move to the *"Establish"* state). The states *"Quit"* and *"Establish"* are recurrent because once the discovery process enters either state the process ends. The other states are transient because if the process is in state "*i*", it may only go to *"Quit"*, the next state, or *"Establish"* and will not reenter state "*i*". Thus, a discovery session converges [73]. Second, we consider that *n* service providers coexist. Since code words in the first message are unique, a service provider may find up to one matched code word. Therefore, there are up to n sessions. Since each session converges, the discovery process converges.■

6.3.4. Hash Results Follow the Uniform Distribution over an Integer Set

In Section 6.2, we assume that the last dozens of bits of one-time code words and one-time secrets follow the uniform distribution over an integer set, respectively. We rely on these assumptions to protect identities and service information from those who do not know the shared secrets. If the one-time secrets do not follow the uniform distribution over an integer set, our encryption method for service information may not be computationally secure. Moreover, if the assumption does not hold the probabilities $p(match \,|\, not\, user)$, $p(not\, user \,|\, match)$, and $p(user)$ are affected. Therefore, exposures may not be predicted well.

The null hypothesis of our first test is that the last dozens of bits in code words follow the uniform distribution over an integer set; and the null hypothesis of our second test is that the last 5 bits of every byte in a one-time secret follow the uniform distribution over an integer set.

We use the chi-square goodness-of-fit test to determine if the data can be adequately modeled by the uniform distribution over an integer set [74]. For the first hypothesis test, there are 2^n possible outcomes (n is the number of bits). For the second hypothesis test, there are 2^5 possible outcomes. To test the hypotheses, we randomly generate a secret that serves as the shared secret. Two bytes of a timestamp and 14 one-byte random numbers are used as a time variant parameter. Next, we use the mechanism discussed in Section 6.2.1 to generate billions of one-time code words and secrets. Then, we count the number of occurrences for each outcome. Last, we calculate the chi-square test statistics and select 5% as the significance level.

Table 6-6 (a) show the first hypothesis test results. We generate 100,000 code words for each *n*, where *n* is from 4 to 13, 25,600,000 code words for each *n*, where *n* is from 14 to 16, and 409,600,000 code words for each *n*, where *n* is from 17 to 20. Only one test (*n*=18) is significant. However, it may be given a false result. Because given the significance level is 5%, it seems reasonable that 1 out of 17 experiments is false. Then, we do 20 experiments to test as *n* equals 18. Table 6-6 (b) shows that only one of the 20 tests is significant (group number 19). Therefore, we do not reject the null hypothesis for *n* from 4 to 20.

For the second hypothesis test, we generate 10,000 one-time secrets. All last 5 bits of the 20 bytes are tested. Only byte number 6 turns out to be significant as shown in Table 6-7 (a). Similarly, we generate 20 groups of 10,000 secrets to test byte number 6 again and no p-value is significant in the tests as shown in Table 6-7 (b). Therefore, we do not reject the null hypothesis.

6.3.5. Performance Measurements

Portable devices such as PDAs and cell phones have good computation capabilities. Those devices are good candidates to aggregate credentials and supply credential automatically for users. Service providers may use various devices for authentication and authorization. Thus, we select the Compaq iPAQs to measure the performance of our protocol. Each PDA has an ARM SA1110 206 MHz processor, 64MB RAM, an expansion pack, and a D-Link DCF-650W wireless card. The wireless cards are set to the 802.11 ad hoc mode and 2Mbps. Our software is developed using Microsoft eMbedded Visual C++ 3.0 and running on Microsoft PocketPC 3.0.

Table 6-6. Test results for code words. (a). χ^2 tests on different length of code words. (b).Revisit the 18th bit of code words. 20 groups of tests with 409,600,000 code words each.

Number of bytes	Number of code words	Possible outcomes	df	χ^2	p-value
4	100,000	16	15	11.3051	0.730682
5	100,000	32	31	20.5702	0.922787
6	100,000	64	63	57.7298	0.664005
7	100,000	128	127	132.405	0.353405
8	100,000	256	255	269.336	0.256982
9	100,000	512	511	474.779	0.87285
10	100,000	1024	1023	1053.64	0.246533
11	100,000	2048	2047	2089.71	0.250371
12	100,000	4096	4095	4171	0.199892
13	100,000	8192	8191	8228.25	0.383691
14	25,600,000	16384	16383	16458.6	0.336991
15	25,600,000	32768	32767	32493.9	0.857056
16	25,600,000	65536	65535	65457.7	0.583852
17	409,600,000	131072	131071	131311	0.31926
18	409,600,000	262144	262143	263525	**0.028311**
19	409,600,000	524288	524287	524770	0.318397
20	409,600,000	1048576	1048575	1050600	0.081073

(a)

Group number	χ^2	p-value
1	262803	0.180973
2	262771	0.192823
3	262940	0.135553
4	262873	0.156687
5	262753	0.199692
6	261747	0.707556
7	261945	0.607420
8	262525	0.298669
9	261892	0.635268
10	261911	0.625357
11	263014	0.114584
12	263307	0.054125
13	262864	0.159683
14	261891	0.635787
15	261482	0.819309
16	262606	0.261093
17	261864	0.649709
18	262326	0.399903
19	263460	**0.034627**
20	262001	0.577392

(b)

Table 6-7. Test results for one-time secrets. (a). χ^2 tests on the 20 bytes of one-time secrets. (b). 20 groups of tests on Byte 6.

Byte Number	χ^2	p-value
0	33.5872	0.343077
1	26.144	0.714396
2	22.3936	0.870264
3	40.2112	0.124346
4	25.344	0.752122
5	33.0048	0.369315
6	47.296	**0.030703**
7	30.7392	0.479408
8	29.6128	0.537351
9	30.2208	0.505915
10	22.4576	0.868118
11	28.4608	0.597293
12	22.6368	0.862004
13	31.1168	0.460338
14	16.4992	0.984558
15	30.7648	0.478109
16	26.4832	0.697865
17	33.2224	0.359397
18	28.7936	0.579982
19	21.9072	0.885906

(a)

Group number	χ^2	p-value
1	26	0.721324
2	22.2949	0.873533
3	37.2115	0.20467
4	24.9679	0.769152
5	37.5385	0.19443
6	37.6474	0.191103
7	36.1471	0.240612
8	23.1987	0.841831
9	26.5192	0.696094
10	26.3654	0.703638
11	25.9551	0.723473
12	21.0064	0.911742
13	27.3077	0.656633
14	26.2372	0.709882
15	43.8846	0.06243
16	37.2732	0.202709
17	25.109	0.76282
18	27.6603	0.638643
19	34.8654	0.289166
20	42.8141	0.076959

(b)

Table 6-8. Performance measurement of the protocol.

Party	Operation	Time
User	Generating 100 one-time code words from domain secrets, and sending discovery messages	55.64ms
User	Waiting time from sending the first message to receiving the first reply	17.64ms
Service provider	Generating code word from the domain secret and verifying all code words and secrets in the first message	4.03ms
Service provider	Generating 50 one-time code words from user secrets	7.68ms
Service provider and user	Each message after the first two messages, (from verifying the code word and service information, sending the message, to the other party receives)	3.58ms

The experimental results show that our protocol is efficient on the PDAs. Table 6-8 shows the measurements of the major procedures. We repeated 100 experiments and calculated the average execution time. When a user generates 100 code words from domain secrets and a service provider generates 50 code words from user secrets, a sophisticated version that generates unique code words as we discussed in Section 6.3.2 is used. The discovery process between a user and a service provider takes up to 100 milliseconds. Each additional service provider involved takes another 30 milliseconds. Therefore, within a reasonable time (a few seconds), a user can finish the discovery process.

6.3.6. Formal Verification

We use BAN logic [65] and an extension [75] to formally verify our protocol. Using the logic's bindings and freshness checking strength, we protect our protocol against attacks, such as message modification and replay attacks. During several rounds of the design and verification process, the logic helps us make design decisions and make our protocol succinct.

Following the convention of the verification process, we convert our protocol to an idealized protocol version as shown in Table 6-9 and we explicitly write the assumptions as shown in Table 6-10. Message A and B in original protocol are not included, because they have only clear text information. The deduction is lengthy, thus we only show stepwise results in Table 6-9. In Table 6-11, we show the deduction of step 1 and omit the similar discussion of the other steps. The results acquired in micro steps 3 and 6 in Table 6-11 are the stepwise results of step 1 in Table 6-9.

Table 6-9. Formal verification using BAN logic.

Step	Idealized Protocol	Stepwise results
1	$\{CB^1_{SUI}, \#(CB^1_{SUI}), \{SRB^1, \#(SRB^1)\}_{KUSI}\}^N$	$S \models CB^1_{SUI}$, $S \models SRB^1$
2	$CB^1_{SUI}, CB^2_{SUI}, \#(CB^2_{SUI}), \{SAB^2, \#(SAB^2)\}_{KUSI}, \{CW^2_{SUL}, \#(CW^2_{SUL})\}^M$	$U \models CB^2_{SUI}$, $U \models SAB^2$, $U \models CW^2_{SUL}$
3	$CW^2_{SUL}, CW^3_{SUL}, \#(CW^3_{SUL}), \{SRB^3, \#(SRB^3)\}_{KUSL}$	$S \models CW^3_{SUL}$, $S \models SRB^3$
4	$CW^4_{SUL}, \#(CB^4_{SUL}), \{SAB^4, \#(SAB^4)\}_{KUSL}$	$U \models CW^4_{SUL}$, $U \models SAB^4$

Table 6-10. Assumptions.

1	$U \models U \xleftrightarrow{K_{USI}} S$, $S \models S \xleftrightarrow{K_{USI}} U$, $U \models U \xleftrightarrow{CB_{USI}} S$, $S \models S \xleftrightarrow{CB_{USI}} U$, $U \models U \xleftrightarrow{DomainSecret_{USI}} S$, $S \models S \xleftrightarrow{DomainSecret_{USI}} U$ $U \models U \xleftrightarrow{K_{USL}} S$, $S \models S \xleftrightarrow{K_{USL}} U$, $U \models U \xleftrightarrow{CW_{USL}} S$, $S \models S \xleftrightarrow{CW_{USL}} U$, $U \models U \xleftrightarrow{UserSecret_{USL}} S$, $S \models S \xleftrightarrow{UserSecret_{USL}} U$
2	$S \models (U \Rightarrow U \xleftrightarrow{K_{USI}} S)$, $S \models (U \Rightarrow CB)$, $U \models (S \Rightarrow U \xleftrightarrow{K_{USI}} S)$, $U \models (S \Rightarrow CB)$ $S \models (U \Rightarrow U \xleftrightarrow{K_{USL}} S)$, $S \models (U \Rightarrow CW)$, $U \models (S \Rightarrow U \xleftrightarrow{K_{USL}} S)$, $U \models (S \Rightarrow CW)$
3	$S \models \#(t_U)$, $S \models \#(R_U)$, $U \models \#(t_U)$, $U \models \#(R_U)$, $S \models \#(t_S)$, $S \models \#(R_S)$, $U \models \#(t_S)$, $U \models \#(R_S)$

Table 6-11. Formal verification of step 1.

Micro steps	Deduction	Notes
1	$\dfrac{S \overset{Domain\ Secret}{\infty} U, S[CB^1]_{Domain\ Secret}}{S \mid\equiv U \mid\sim CB^1}$	Using Postulates 1 in [75]. The service provider finds a match in a code word. This only means that there is a possibility that the code word generating party knows the secret.
2	$\dfrac{S \mid\equiv U \mid\sim CB^1, \#(CB^1)}{S \mid\equiv U \mid\equiv CB^1}$	Using Postulates 2 in [75]. The service provider believes that the code word bits are recent.
3	$\dfrac{S \mid\equiv U \mid\equiv CB^1, U \Rightarrow CB^1}{S \mid\equiv CB^1}$	Using Postulates 3 in [75]. The service provider believes the code word bits.
4	$\dfrac{S \triangleleft \{SRB^1\}_{KSUI}, S \mid\equiv U \overset{KSUI}{\rightarrow} S}{S \mid\equiv U \mid\sim SRB^1}$	Using the message-meaning rule in [65]. The service provider believes the user once said the service request bits.
5	$\dfrac{S \mid\equiv U \mid\sim SRB^1, \#(SRB^1)}{S \mid\equiv U \mid\equiv SRB^1}$	Using the nonce-verification rule in [65]. The service provider believes that the service request is recent and the user still believes in it.
6	$\dfrac{S \mid\equiv U \mid\equiv SRB^1, U \Rightarrow SRB^1}{S \mid\equiv SRB^1}$	Using the jurisdiction rule in [65]. The service provider believes the service request bits.

6.3.7. Threat Analysis and Discussion

Without knowledge of secrets, external attackers do not understand code words and service information. The chances of guessing the correct code words and service information are low as we discussed in Section 6.3.1.

Users and service providers who know the shared secrets might be malicious. With knowledge of the secrets, internal attackers might be more destructive. However, when a service provider assigns privileges to individual users, a user only discovers what he is authorized to discover. On the other hand, a service provider might cheat a user and learn the user's identity and the service request, but cheating may not be easily achieved. To learn a user's identity, a service provider also exposes his own identity. Our protocol ensures that the more confident one is of the user's identity the more exposure is required of the service provider's identity. To learn a user's service request, given the number of existing service types is large, a service provider needs to exchange many rounds of messages. The service provider may claim that he has all types of services. Nevertheless, if a user suspects that a service provider is cheating, he may use the wild card search (widely available in service discovery protocols) to find all service types that the service provider provides to him. A comparison of service information in the interactive messages and the wild card search tells whether the service provider cheats.

The number of service providers that a user interacts and the number of users that a service provider has may become signatures and thus reveal the identities in the first and second messages, respectively. For example, Bob interacts with 29 service providers. When a message with 29 code words is set, it is likely that Bob is discovering services. To prevent the exposures, random numbers that do not conflict with code words may be inserted. One strategy is to insert a different amount of random numbers as code words in different messages. Thus, a user or a service provider looks different in each discovery. Another strategy is to insert random numbers to reach the same amount of code words in a message for certain number of bits in a code word. For example, from Table 6-3, all users that interact with 19 to 36 service providers use 7 bits in code words. For those users, they all specify 36 code words including random numbers. For example, if a user interacts with 25 service providers, he adds 11 random numbers. Therefore, many users look the same. Similarly, service providers may insert random numbers to obscure the number of users that they have.

6.4. Summary

In this chapter, we identified that involving only the necessary users and service providers for service discovery in pervasive computing environments may be as difficult as a chicken-and-egg problem. We designed a progressive and probabilistic approach to protect sensitive information effectively and preserve privacy for users and service providers. Users and service providers in turn expose bits of information to determine whether further exposure is necessary. We analyze the mathematical properties, and via experimentation and hypotheses tests we demonstrate that security properties hold. Formal verification and threats analysis suggest that our protocol is sound. Performance measurements show that our protocol runs efficiently in most cases on devices such as PDAs.

Our approach has its limitation in extreme cases. The progressive is not efficient when many users and service providers present in a place, for example, a stadium. A discovery process will cause many false positive matches and waste of computation resources and energy. A possible solution is to integrate with more precise discovery approaches, such that users and service providers can eliminate most unnecessary exposures in the first several messages. However, the rules to determine when using precise approach and when using progressive approach could be complex. Moreover, our approach is not effective when the possible user set is small. For example, if Bob is the only person that has access to a room, then one can tell Bob is in the room when any discovery message is sent from the room without need to understand the code words.

The calculation of $p(user)$ and $p(service)$ is based on history information, and we are working on providing detailed algorithms for updating them. Service providers, who move around frequently, may only use recent discovery information for the calculation, while less mobile service providers may use more history information for the calculation.

In addition, our current approach does not support discovery by service attributes. We are evaluating a data structure to express multiple service attributes as a hash result. The challenge is to design an algorithm to express available services progressively and an efficient algorithm to update each user's available service types in the hash form.

Under the current model, exposures are measured in probability and considered the same privacy risk if the probabilities are the same. In reality, exposing certain service information might be more serious than others, for instance exposing a medical device verses exposing an MP3 player that a person carries. However, quantifying the privacy risk may be difficult.

CHAPTER 7. PRIVATE ENTITY AUTHENTICATION FOR PERVASIVE COMPUTING ENVIRONMENTS

7.1. Introduction

In this chapter, we propose *the Master Key,* which is a novel entity authentication model for pervasive computing environments. The model is based on the PrudentExposure model that we discussed in Chapter 5 and the progressive exposure model that we discussed in Chapter 6. Nevertheless, the new security model has its application beyond secure service discovery.

We prove our identities everyday by showing the possession of access tokens. Using a key to open a lock may be the most common form, which has about 4000 years of history since ancient Egypt [76]. As one may access many locks, traditional master keys were designed to enable accessing multiple locks with a single key. Nevertheless, master keys are not widely used. Instead, people carry multiple access tokens for entity authentications, for example, keys, magnetic stripe cards, smart cards, RFID tags, and other tokens. The Master Key has the advantages of master keys and multiple access tokens.

Traditional master keys are convenient. One does not need to carry many keys and memorize relationships between keys and locks. However, traditional master keys have fatal problems that are not suitable for everyone's daily usage. For example, the delegation of a master key equals delegating access to all locks that one has privilege to access. Revocation of the privilege of accessing one lock from a master key is costly because the lock and the keys of other owners need to be replaced. In addition, if an intruder acquires a master key, then the intruder may open many locks. Moreover, locks that support master keys are vulnerable to the malicious insider who has a normal key [77].

The use of multiple access tokens does not have the fatal delegation and revocation problems as traditional master keys have because one token usually matches one lock. If a key-lock pair is compromised, it does not put other locks at risk. Issues of delegation and revocation are better addressed by replacing keys with modern access tokens, for instance, a hotel room key in the form of a magnetic stripe card or a smart card. With the encoding of privileges within a digital form, the delegation and revocation of the privileges are done on the computers at the front desks. Moreover, modern access tokens improve usability in a wide variety of applications, for example, unlocking a car using a remote control; accessing an enterprise facility using a smart card badge; entering a parking facility using a RFID gate card; opening a hotel room using a magnetic stripe card; or locking and unlocking a computer by wearing a token [56]. Additional token designs are emerging as well as their applications. Nevertheless, the management of access tokens and memorizing the token-lock relationships become inconvenient and difficult as the number of tokens increases.

In pervasive computing environments, entity authentications might be ubiquitously necessary. An intuitive question is how to achieve both the advantages of traditional master keys and multiple access tokens while avoiding their disadvantages. The *Master Key* that we propose aggregates the digital forms of all access tokens that its owner has. The tokens on the Master Key and their respective locks maintain their original relationships (one token matches one lock). Therefore, the advantages of using multiple access tokens are retained. The goal of this paper is to design an approach that properly selects access tokens for entity authentications and therefore achieves the convenience of traditional master keys. From now on, we refer to keys as the digital access tokens and locks as the target resources.

The Master Key is invoked when the owner pushes a lock or unlock button. At that time, the Master Key discovers the right key for a lock and authenticates with the lock.

7.2. The Master Key Design

In our design, the Master Key is active (see Section 2.7 in Chapter 2). It initiates an authentication process by sending a broadcast message together with a challenge, as shown in the left half of Figure 7-1. Since the Master Key aggregates digital keys of a user, it needs to find the right key to operate a lock. Therefore, it queries a set of locks for which it has keys. If a lock does not find itself in the set, it keeps silent. Otherwise, the lock responds and sends its challenge. Then the Master Key responds and supplies a key to operate the lock. (Perhaps there are multiple locks that reply back. We will discuss the situation in Section 7.4.) If any response is not correct, the Master Key or the lock terminates the authentication process.

The Master Key can be passive when a lock and the Master Key touch each other. In such a case, the Master Key and the lock communicate over a physical link. Thus, we may use the same approach as the active Master Key. In the rest of this paper, we only discuss the case that the Master Key and a lock communicate via a wireless link.

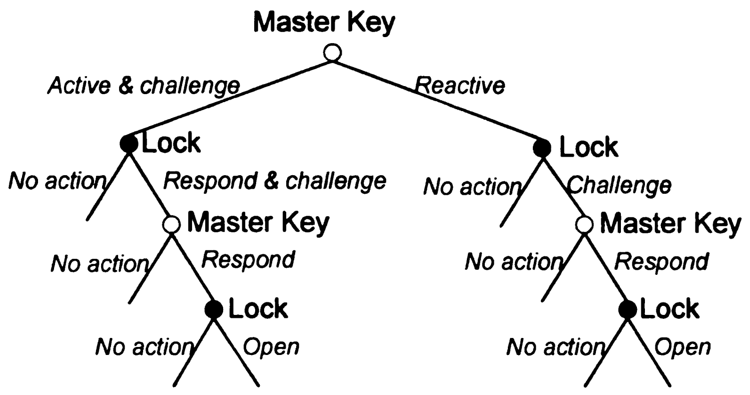

Figure 7-1. Reactive and active Master Key Design.

We do not select reactive as the Master Key's initial status. If the Master Key is reactive, it waits for locks to send challenges (shown in right half of Figure 7-1). When a lock sends a challenge together with its identity, the Master Key knows which key to use. Thus, the design of the Master Key may be simpler. However, as we discussed in Section 2.7, a reactive key may be maliciously accessed without its owner's knowledge. In addition, the reactive Master Key requires all the locks to frequently send challenges. Locks that use batteries such as Remote Keyless Entry systems on cars may not be able to afford the expense of frequently sending messages.

Unlike the existing keys and locks that expose their information in full whenever they are used, a key on the Master Key and a lock may select the degree of exposure during the challenges and responses. This flexibility enables a key and a lock to decide when to expose and how much to expose. In addition, partial exposure is very useful when discovering locks. If a lock or a key is not present, we can stop further exposure and avoid unnecessary full exposure. We will discuss the details in Section 7.2.3 and Section 7.2.4. In this paper, we focus on the case that the Master Key and a lock mutually authenticate each other in three messages, as shown in the left half of Figure 7-1. A key and a lock may exchange partial information in many rounds, and detailed discussion of such interactions may be found in [78].

7.2.1. Protect Privacy Information from Outsiders – Private Authentication

Unlike standard authentication protocols that send identifications in clear text, our authentication process exchanges only code words. A code word is generated at both the Master Key side and a lock's side. It is calculated from a shared secret between the Master Key and lock. One sends a code word for authentication, while the other verifies. Since only a party who knows the shared secret understands a code word, identifications are protected and not exposed to outsiders. Thus, private authentication is achieved.

Code words in our design have two formats: the hash format and the Bloom filter format. The hash format is used when the key and lock have a one-to-one relation, for example after the Master Key has discovered the target lock. The Bloom filter format is used when keys and locks have one-to-many relationships. For instance, the Master Key queries a set of potential locks, or a lock needs to identify a key owner among many key owners.

The top half of Figure 7-2 illustrates the generation of a code word in the hash format. A time variant parameter (TVP) and the shared secret are the two inputs to *h(*)*, which is the hash-based message authentication codes proposed in [71]. A TVP consists of a timestamp and a random number. The Hash function, MD5 or SHA-1, is used in the place of *h()*. The hash result is the hash format code word. The preimage resistance and collision resistance properties of MD5 and SHA-1 ensure that it is computationally difficult to find the shared secret from the hash result [58]. The TVP and hash result are sent to the other party for verification.

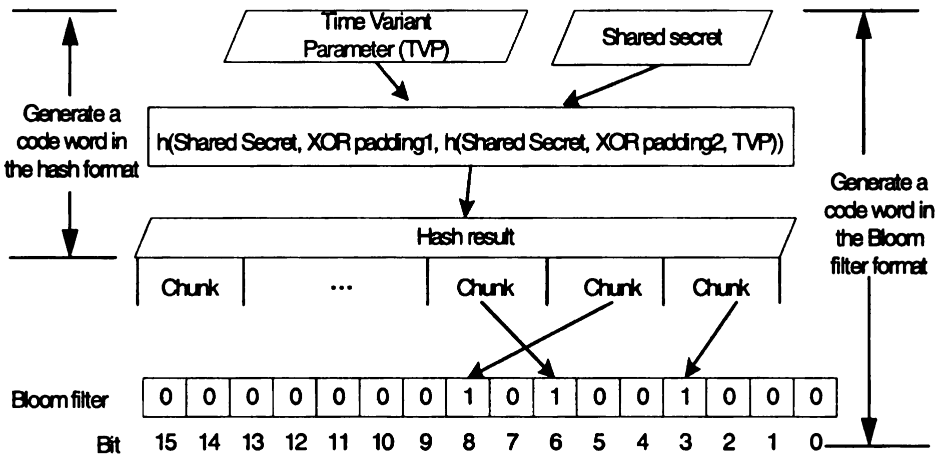
Time Variant Parameter (TVP)
Shared secret
Generate a code word in the hash format
h(Shared Secret, XOR padding1, h(Shared Secret, XOR padding2, TVP))
Hash result
Chunk
...
Chunk
Chunk
Chunk
Generate a code word in the Bloom filter format
Bloom filter
0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 0
Bit
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Figure 7-2. The generation of a code word.

For a code word in the Bloom filter format, a hash result as we discussed above is first generated. Then, the hash result is further separated into chunks as shown in Figure 7-2. The size of the chunks depends on the length of the Bloom filter, which is an array of 2^X bits. (To be fit within a network packet, a Bloom filter is equal or less than $2^{13} = 8192$ bits.) If the Bloom filter is 2^{13} bits, then the chunk size is 13 bits. Since the hash result is 128 bits for MD5 and 160 bits for SHA-1, a hash result is separated into at least 10 chunks. All bits in a Bloom filter are initially set to zero. The value of a chunk serves as an index to a Bloom filter and the corresponding bit is set to one. A code word is represented by a combination of several bits that are indexed by the chunks. For example, in Figure 7-2, bits 8, 6, and 3 represent a code word.

For all potential locks that the Master Key wants to discover, it repeats the above process using the same TVP and sets the code word bits in the same Bloom filter. Then, the Bloom filter and the TVP are broadcasted to query the locks in the vicinity.

If a lock and each of its key owners share a unique secret, the lock may generate code words for all its key owners in a Bloom filter via the same process. Then, the lock sends the Bloom filter and requests the key owner who sends the discovery message to identify himself.

The probability of finding a hash result from a Bloom filter is $1/(\frac{m!}{(m-k)!})$, where k is a code word length and m is the number of bits set in a Bloom filter. The denominator is the number of permutations of k bits from m bits (that is, select k bits from m bits and then make arrangements of the k chunks to guess a hash result). Only part of the hash result might be found if the code word is generated by part of the chunks. Even if the hash result is found, it is still computationally difficult to find the shared secret as in the situation of the hash format code words.

The Bloom filter format of code words is scalable. Hundreds of code words may be expressed in a Bloom filter. For instance, if the Master Key uses a 2^{13}-bit Bloom filter, of which 50% are set, and on average each code word is 5 bits, then at least $2^{13}\times 50\% \div 5 = 819$ code words may be set in a discovery message. The result (819 code words) is a lower bound, which is calculated from the extreme case that no two code words set the same bit.

Code word verification in the Bloom filter format is efficient and independent of the number of code words in a Bloom filter. A party calculates the hash results, as we discussed above, and then verifies whether the bits indexed by the chunks in a Bloom filter are set to one. If any bit is not one, then the code word does not match. A property of Bloom filters ensures that if two hash results match, the Bloom filter format of the code words match [19]. It is possible that a party may find false positive matches, and we will discuss the probability of false positive cases in Section 7.2.3 and the relative overhead in Section 7.3.1.

7.2.2. Secure the Authentication Process

Figure 7-3 shows the generic protocol for the authentication process. In the first and second messages, both the Master Key and a lock post challenges, TVPs, respectively; while in the second and third messages the other parties respond with code words based on the challenges. Thus, mutual authentication is attained.

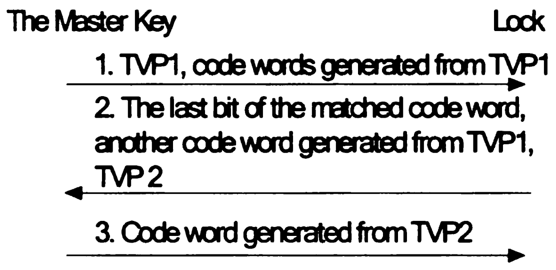
The Master Key
Lock
1. TVP1, code words generated from TVP1
2. The last bit of the matched code word, another code word generated from TVP1, TVP 2
3. Code word generated from TVP2

Figure 7-3. The generic interaction protocol shows the exchange of code words between the Master Key and a lock.

The code words are one-time code words. Based on TVPs, code words between the Master Key and a lock differ between protocol runs. In addition, a replay code word can be easily detected by checking the freshness of a TVP. (The clocks on the Master Key and locks are required to be loosely synchronized.)

The total number of bits set in a Bloom filter may be maliciously used as a signature to track or identify a Master Key owner. To counter the attack, code words and some random bits are mixed together to reach a fixed ratio of the number of bits set and total number of bits in a Bloom filter. Thus, all Bloom filters look the same. Moreover, code word lengths in the Bloom filter format are obscured. A lock only indicates the last bit of the matched code word in its reply message, as shown in Figure 7-3. Thus, every code word looks the same to an attacker.

7.2.3. Protect Privacy Information from Insiders

Since the authentication process includes discovery of the locks and keys, unnecessary code words may be transmitted. Although the Master Key and locks exchange code words, insiders understand the code words. In some situations there are privacy concerns among insiders. For example, Bob does not want other gym key owners to use the knowledge of the key for the gym to identity or track him at places other than the gym. To address the problem, Bob's Master Key may speak a partial code word for the gym door lock in a discovery message. A partial code word causes an insider to be uncertain whether Bob has the gym key. If Bob is not near the gym door, his Master Key will not receive a reply message from the gym door lock, and thus Bob preserves his privacy. A partial code word increases the number of false positive cases, which for example, the gym's door lock may have communication and computation overhead to interact with some illegitimate key owners. (Illegitimate key owners do not gain access because the code words in the last messages can only be generated by legitimate key owners.)

There might be no privacy concerns among insiders. For example, Bob and his wife Alice are not concerned that their Master Keys speak code words for their cars, because only their Master Keys and cars understand the code words. When there is no privacy concern, a precise code word should be used to avoid unnecessary communication and computation overhead. For example, Bob's Master Key always speaks a precise code word for his car.

The Bloom filter has the properties that meet our two needs: protecting privacy and avoiding unnecessary overhead. Both precise code words with low false positive rates and partial code words with high false positive rates may be specified in the same Bloom filter.

We examine some mathematical properties of the Bloom filter for our case. One may find some formal mathematical analysis of the false positive rates and the calculation of the expected false positive rate in [72, 79]. Unlike the analyses, which are based on having each element in a Bloom filter with the same length, the Master Key may use various lengths for different code words. Moreover, the Master Key uses a fixed ratio between the numbers of bits set and the Bloom filter lengths. The false positive rate in our case is a typical sampling with replacement problem. Therefore, the false positive rate at a lock's side is:

$$p(\textit{code word match} \mid \textit{not key owner}) = \left(\frac{m}{n}\right)^{\mathrm{k}} \qquad (1)$$

where n is the Bloom filter length, m is the number of bits set in the Bloom filter, and k is the length of a code word. Thus, given an illegitimate key owner, the false positive rate depends on n, m, and k. (The analysis of the false positive rate at the Master Key's side is similar, and thus we omit it.)

Examining equation 1, we find the property that is useful in our case. When the length of a code word increases and the m/n ratio is fixed, the false positive rate decreases as shown in Figure 7-4. For example, if the m/n ratio is at 50%, a code word of 1 bit has a false positive rate of 0.5, a code word of 5 bits has a false positive rate about 0.03, and a code word of 10 bits has a false positive rate less than 0.001. The m/n ratio at around 50% provides a good span of false positive rates for the code words from 1 bit to 10 bits. By default, 50% of the m/n ratio is used. In summary, if a key-lock pair wants to achieve a low false positive rate, they may exchange more bits (say 10 bits) in a code word. If there are privacy concerns, they may exchange fewer bits (1, 2 or 3 bits) of a code word.

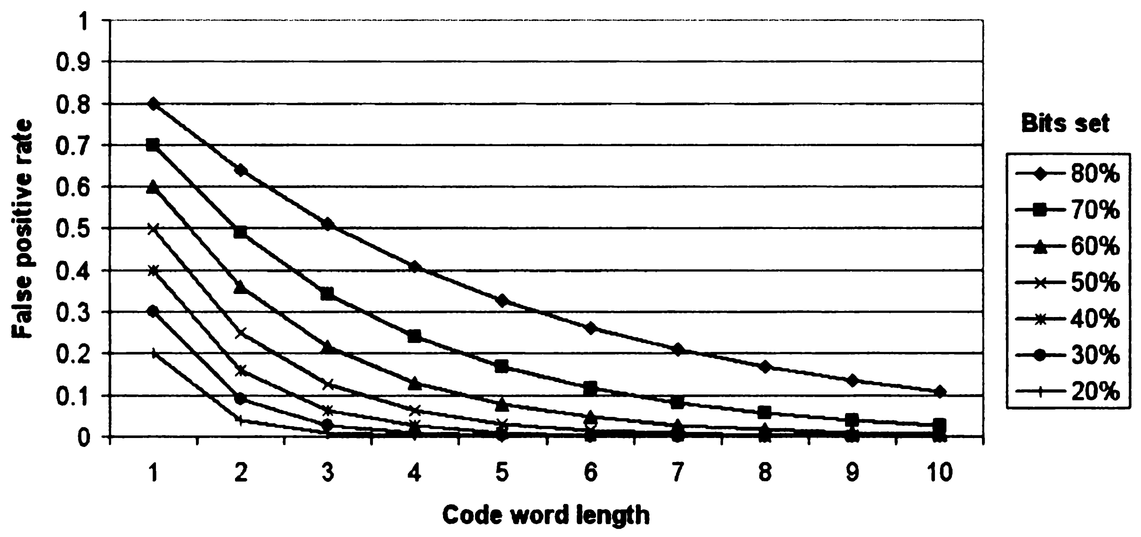

Figure 7-4. False positive rates decreases as the length of the code word increases.

Long code words reduce unnecessary communication and processing overhead. To further reduce unnecessary overhead, another hash algorithm or another secret may be used to generate a code word of 20 bits or longer. Very long code words are useful for applications that require extremely low false positive cases, such as for Remote Keyless Entry systems.

The use of partial code words changes the order from having the key owner first expose code words precisely to having locks first expose code words precisely. Precise exposure at a later time has the advantage when privacy is a concern. If there is a mismatch in the code word, a party that exposes later may avoid the exposure. If both a key owner and a lock have privacy concerns, the issue becomes a "chicken-and-egg problem". The progressive approach proposed in [78] may be used.

7.2.4. The Master Key Protocols

In this section, we present three protocols to demonstrate that our approach is flexible for different types of key-lock relations. A *unique key* is one that may be owned by one or a few owners to open a lock, for example a car's Remote Keyless Entry system; a *group key* is one that a lock is able to authenticate as a key, but key owners are not differentiable, for example a gate card to entering a parking facility; and an *individual key* is one for which a specific key owner among a group can be identified, for example a badge for entering a factory.

The initialization processes for all relationships are the same. Shared secrets are delivered from locks to the Master Keys via secure channels. In addition, locks indicate the number of bits of the code words in the Bloom filter format.

All protocols have the same first message, which the Master Key uses to discover potential locks. Thus, we discuss the first message only in the protocol for unique keys. In the next two messages, the Master Key and a lock authenticate each other using their respective authentication protocol.

7.2.4.1 The Unique Key

The Master Key initiates an authentication process by broadcasting a Bloom filter format of code words with a TVP as shown in Figure 7-5(a). A lock replies only if it finds a match. The lock indicates the last bit of the code word, and uses another hash algorithm to generate a code word in the hash format. In addition, the lock posts its challenge, the TVP. By default, MD5 is used for the code words in the Bloom filter format, while SHA-1 is used for the code words in the hash format. If the Master Key finds the indicated last bit of the code word in the Bloom filter format and the hash format code word are correct and match, it replies back with another hash format code word. It is possible that the Master Key may have several Bloom filter format code words that have the same last bit. However, comparison of the code words in the hash format will exclude the false positive cases.

Notation:
L is a lock. M is the Master Key.
t_X is a timestamp that X attaches.
R_X is a random number that X generates.
A t_X and a R_X is a TVP.
$(\ )_{KX}^{-1}$ is X's signature using its signing private key.
$BF_P(y, S)$ is a code word in a Bloom filter that P generates from a shared secret, S, and a TVP, y.
$Hash_P(y, S)$ is a code word in the hash format that P generates from a shared secret, S, and a TVP, y.
MB_P is the last bit of a code word that a party, P, finds the match.

Msg No.	Sndr/Rcvr	Message
1	$M \rightarrow L$:	$BF_M(R_M, t_M, S_{Unique}), R_M, t_M$
2	$L \rightarrow M$:	$R_M, t_M, MB_L, R_L, t_L, Hash_L(R_M, t_M, S_{Unique})$
3	$M \rightarrow L$:	$R_L, t_L, Hash_M(R_L, t_L, S_{Unique})$

(a). The protocol for the unique key.

Msg No.	Sndr/Rcvr	Message
1	$M \rightarrow L$:	$BF_M(R_M, t_M, S_{PlainText}), R_M, t_M$
2	$L \rightarrow M$:	$R_M, t_M, MB_L, R_L, t_L, Hash_L(R_M, t_M, S_{Group})$
3	$M \rightarrow L$:	$R_L, t_L, Hash_M(R_L, t_L, S_{Group})$

(b). The protocol for the group key.

Msg No.	Sndr/Rcvr	Message
1	$M \rightarrow L$:	$BF_M(R_M, t_M, S_{domain}), R_M, t_M,$
2	$L \rightarrow M$:	$R_M, t_M, MB_L, Hash_L(R_M, t_M, S_{domain}),$ $(Hash_L(R_M, t_M, S_{domain}))_{KL}^{-1}, BF_L(R_L, t_L, S_{individual}), R_L, t_L$
3	$M \rightarrow L$:	$R_L, t_L, MB_M, Hash_M(R_L, t_L, S_{individual})$

(c). The protocol for the individual key.

Figure 7-5. The Master Key protocols.

If there are several key owners, they share the same secret with the lock. Usually, this type of key-lock relation is used for owners and locks without privacy concerns, and thus a 10-bit code word in the Bloom filter format is used. However, if key owners have privacy concerns, they may use a code word with fewer bits. The protocol remains the same, while the number of bits of a code word changes. Alternatively, the individual key type may be used to address key owners' privacy concerns.

7.2.4.2 The Group Key

The special requirement for the group key is that a lock should not be able to differentiate key owners from their keys. This means that all owners should have the same key. The same protocol as the unique key may be used, since the only difference between the group key and the unique may be the number of key owners. But a key owner should only speak a 1-bit or 2-bit code word. A short code word ensures that a lock cannot differentiate among key owners. Because when there are two different code words of length 1 or 2 bits, it is very likely that the lock will find a false positive match and a true match in the first message. The key owner is able to tell that he has a different key than other owners when the lock replies with an incorrect code word in the hash format.

If the overhead caused by the false positive cases is large, a lock and its owners may use more bits for the code word in the Bloom filter format. However, the code word is generated from some plain text, as shown in Figure 7-5(b). The plain text in the message may be some human readable text such as "the CS department's mail room" or "XYZ Company's parking lot". The use of plain text instead of a secret changes the order of who first expresses knowledge of a secret. Since the lock expresses its knowledge first, the Master Key knows that it shares the same secret as other key owners. If there is more than one secret, the lock may provide an incorrect code word in the second message. Moreover, the plain text is easy for key owners to verify that code words in the Bloom filter format are generated from some meaningful and reasonable text.

7.2.4.3 The Individual Key

A key owner of the individual key can be identified among the group of key owners. A secret that the lock shares with a key owner is different from the secrets that it shares with other key owners. Thus, each key-lock pair has an individual secret. However, the individual secret is not used to generate the Bloom filter format of a code word in the first message. Since the Master Key may specify many code words in the first message and a lock may have many key owners, many false positive cases may happen. To address such a situation, another domain secret is shared among all key owners and is used in the first message to identify the lock. Figure 7-5(c) shows the protocol for the individual key.

In the reply message, the lock proves its knowledge of the domain secret in the hash format. If the key owners are concerned that the reply message may come from another key owner who impersonates the lock, a digital signature may be used in place to counter the attack. As shown in Figure 7-5(c), the lock signs the hash format code word.

The lock sends another Bloom filter in the second message that encodes code words for every key owner. The code words are generated from the individual secrets. Furthermore, the lock may set some random bits in the Bloom filter to hide the number of key owners that it has. In the third message, the Master Key indicates its identity by specifying the matched code word in the Bloom filter and proves its knowledge of the individual secret.

Revocation methods are different for the three key types. Invalidating the unique secret revokes a unique key. To revoke an individual key from a key owner, a lock invalidates the individual secret, while notification of a new domain secret to other key owners may not be imminent. However, to revoke a group key from a key owner, all other key owners need to update their group keys. This is an open problem in our solution. If an owner updates his key when he finds that the key has expired, the lock system may be able to determine the owner's identity because he has just updated his key. We are designing an approach, which is similar to sending email to a group of recipients, so that a new group key is dispatched to all key owners at the same time.

7.3. System Analysis and Evaluation

In this section, the effectiveness of privacy protection against insiders and its relative overhead are discussed. Performance measurement of our protocols shows our approach is efficient. Formal verification of the security and privacy properties is also illustrated.

7.3.1. Analysis of Privacy Protection against Insiders and its Overhead

An insider recognizes a code word in the Bloom filter format, but whether the code word is from a true key owner is a probability, $p(key\ owner \mid match)$ (because there are false positive cases). This probability may be calculated from equation 2, where $p(key\ owner)$ is the percentage of key owners among all people who send discovery messages at a place, and $p(match \mid not\ key\ owner)$ is the false positive rate of a code word in the Bloom filter format. $p(match \mid key\ owner)$ is one because there is no false negative case for code words in the Bloom filter format. The numerator and the denominator on the right side of equation 2 are $p(key\ owner, match)$ and $p(match)$, respectively.

$$p(key\ owner \mid match) = \frac{A \times B}{C \times D + A \times B} \qquad (2)$$

where

A = $p(match \mid key\ owner)$,

B = $p(key\ owner)$,

$C = p(match \mid not\ key\ owner)$,

$D = p(not\ key\ owner)$.

Figure 7-6 illustrates the relation between the number of bits in a code word and $p(key\ owner \mid match)$ for various $p(key\ owner)$ values. The false positive rates, $p(match \mid not\ key\ owner)$, are based on setting 50% of the bits in the Bloom filters. Note $p(key\ owner \mid match)$ is for one lock at a place. Different locks may have different $p(key\ owner \mid match)$ values at the same place.

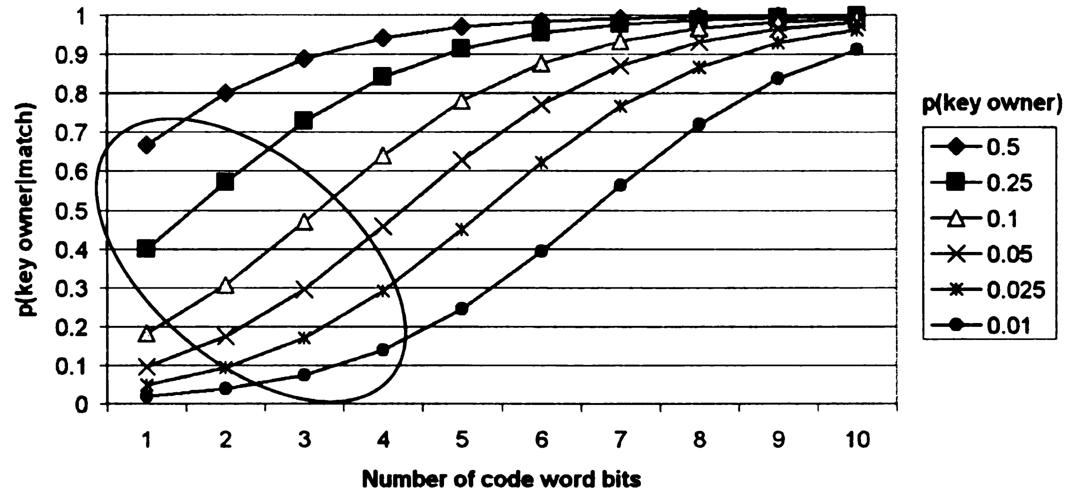

Figure 7-6. The relation between the number of bits in a Bloom filter format code word and $p(key\ owner \mid match)$.

The eclipse area in Figure 7-6 shows the number of bits that a key-lock pair uses to protect privacy among insiders. A lock may count successful and unsuccessful protocol runs (unsuccessful runs include false positive matched and unmatched code words in Bloom filters that the lock hears), and then calculate $p(key\ owner)$ over a period of time at its place. Nevertheless, at other places that no successful run happens, $p(key\ owner)$ is unknown to insiders. An insider is uncertain when he finds a match in the Bloom filter format of a code word because the $p(key\ owner \mid match)$ falls within a wide range. Even if $p(key\ owner)$ is known, $p(key\ owner \mid match)$ is a probability that is not significantly true.

The false positive overhead at the lock's side is $1 - p(key\ owner \mid match)$. Thus, the overhead is high if few bits are used for a code word, while the overhead for a 10-bit code word is low. A lock may calculate $p(key\ owner)$ over a period of time. If the overhead is a concern, it may notify its key owners to adjust the length of a code word. When a lock is not in the vicinity, the overhead at the Master Key side includes the calculation of a code word in the Bloom filter format and possibly the verification of the hash format code word in a false positive case. We will show in the next section that the overhead in terms of processing time is small.

7.3.2. Performance Measurement

Handheld devices such as cell phones or PDAs are good candidates for Master Keys, since people regularly carry them. Locks may have diverse processing and communication capabilities. Some may have limited processing power, while others may support hundreds of key owners. We chose to measure our implementation on PDAs. A Compaq iPAQs with an ARM SA1110 206 MHz processor, 64MB RAM, and a D-Link DCF-650W wireless card runs as the Master Key. A Dell AXIM X5 with Intel PX250 400 MHz processor, 64MB RAM, and a Dell TrueMobile 1180 wireless card runs as a lock. The PDAs run Microsoft PocketPC 3.0, and the wireless cards are set to 2Mbps in the 802.11 ad hoc mode.

Table 7-1 shows processing and communication times of the major components of the Master Key and lock. The processing time is an average measurement of 100 runs. It takes about 231 ms for the Master Key to generate and send the first message with 820 code words. The lock takes about 4 ms to generate and send the second message, and the Master Key takes about 3 ms to generate and send the third message. In case of the individual key, it takes the lock another 169 ms to generate a Bloom filter with 500 key owners specified. One protocol run takes less than a half second in this extreme case, in which a person specifies 820 code words and a lock has 500 key owners. Therefore, our design is efficient in most cases.

Table 7-1. Performance measurement of the major protocol components.

Party	Operation	Time
The Master Key	Generate a 2^{13}-bit Bloom filter with 820 code words. (Average code word length is 5 bits.)	210.86 ms
The Master Key	Generate and set random bits in the Bloom filter to reach 50% of bits set.	12.83 ms
The Master Key	Send the first message.	7.63 ms
Lock	Generate and check the code word in the Bloom filter format in the first message.	<1 ms
Lock	Generate a code word in the hash format.	< 1 ms
Lock	Send the second message.	3.13 ms
The Master Key	Waiting time for the reply message.	11.4 ms
The Master Key	Verify the code word in the hash format from the lock and generate another code word in the hash format.	<1 ms
The Master Key	Send the third message.	2.04 ms
Lock	Generate a 2^{13}-bit Bloom filter with 500 key owners. Each key owner is identified by 2 bits.	168.76 ms
The Master Key	Generate a code word from an individual secret and check it in a Bloom filter.	< 1 ms

7.3.3. Formal Verification of the Protocols

We use BAN logic [65] to verify our protocols to assure that the bindings are correct and messages are fresh. An extension is added to the logic for the verification of code words in the Bloom filter and hash formats. We add three logic constructs:

$P \overset{Y}{\infty} Q$	P shares a secret Y with Q. Q may be an individual or a group.
$(M \subset G)$	M is a member of group G.
$P[CW]Y$	P finds a match in a code word, CW, which uses Y as a secret. If the code word is in the Bloom filter format, there is a possibility that the party that generates the Bloom filter knows the secret Y, as we discussed in Section 7.2.3 and 7.3.1. If the code word is in the hash format, the party that generates the code word knows the secret.

The message-meaning and nonce-verification rules are extended as follows:

$\dfrac{P \overset{Y}{\infty} Q, P[CW]Y}{P \models Q \mid\sim CW}$	When P finds a match of a code word, P knows that Q once said it. If Q is a member of a group, P only knows that one member (including P) once said the code word. If the code word is in the hash format, P is sure that Q said it. If the code is in the Bloom filter format, P knows there is a probability that Q said it.
$\dfrac{P \models Q \mid\sim CW, \#(CW)}{P \models Q \models CW}$	Based on the freshness of the code word, P believes that the code word is fresh.

Since the verifications of the protocols are similar, we only discuss the protocol for the individual key type. Following the BAN logic's procedure, we convert the protocol to an idealized protocol as shown in Table 7-2 (a). Then we explicitly write our assumptions in Table 7-2 (b). Next, we deduct step-by-step to reach conclusions and check whether the conclusions are consistent with our expectation. Due to space limitations, we omit the lengthy deduction of the protocol and only show the stepwise results in Table 7-2 (a). We reach the conclusions: a lock believes that the Master Key has the key to operate it, and the Master Key believes that the lock is authentic.

Table 7-2. Formal verification of the protocol for the individual key.

Msg no.	Idealize protocol	Stepwise results
1	$BF_M\{R_M,t_M\}$	$L\models(M\subset G)\| \equiv BF_M$
2	$Hash_L\{R_M,t_M\},Hash_L\{R_L,t_L\}\}_{KL^{-1}}$, $BF_L\{R_L,t_L\}$	$M\models L\models Hash_{Sdomain}$, $M\models L\models BF_L$
3	$Hash_M\{R_L,t_L\}$,	$L\models M\models Hash_{Sindividual}$

(a). The idealize protocol of the individual key and the stepwise results of our verification.

$M \models M \xleftrightarrow{Sindividual} L$, $L \models M \xleftrightarrow{Sindividual} L$, $M \models M \xleftrightarrow{Sdomain} L$, $L \models M \xleftrightarrow{Sdomain} L$
$L\models\#(t_L)$, $M\models\#(t_L)$, $L\models\#(t_M)$, $M\models\#(t_M)$, $L\models\#(R_L)$, $M\models\#(R_L)$, $L\models\#(R_M)$, $M\models\#(R_M)$, $M \models \xrightarrow{K} L$

(b). Assumptions.

7.4. Discussion

During our design of the Master Key, an interesting question was raised: when there are multiple locks at a place and the Master Key owner has the privileges to access them, to which lock should the Master Key send the operation code? For example, suppose Bob walks to his office and pushes a button on the Master Key to unlock the door. However, the mailroom is also close by. Which door should the Master Key unlock? If the antenna on the Master Key is directional, then the door at which the Master Key points is unlocked. Nevertheless, if Bob also uses the Master Key to unlock his computer in the office, then the office door and the computer may in the same direction. Should both the computer and the door be unlocked? It may be convenient that both the door and the computer are unlocked. Or perhaps the Master Key may store a rule such that unless the office door is unlocked, the computer will not be unlocked. Alternatively, the

Master Key may utilize location or proximity information if such information is available. Only if the Master Key is within a certain distance, the computer accepts the unlock command. For a greater challenge, if Bob has two cars parked side by side outside his house, which car should be unlocked when Bob pushes the unlock button on his Master Key? The Master Key may list the locks that reply back in the order of the frequency of the locks' usage, and then let Bob select one. (Note that Bob selects a lock and not a key, and therefore he does not need to remember key-lock relations.) In short, we may address the problem with proper user interfaces to interact with a Master Key owner or proper integration of information about an ambient environment. These are interesting research issues, but they are out of the scope of this paper.

The Master Key protocols that we discussed so far are susceptible to the Mafia fraud attack, as are many entity authentication protocols. In our case, for example, an adversary may put a device near Bob's office and a device near Bob's house. When Bob pushes a button on the Master Key to unlock his office door, the adversary's devices relays messages: the first discovery message to Bob's house, the house's reply message back to the Master Key, and the Master Key's unlock command to Bob's house door to gain access to Bob's house. If the Master Key notifies Bob that two doors are ready to unlock and lets Bob select a door to unlock before it sends the third message, Bob may notice that something abnormal is occurring. However, if the party that Bob authenticates with is malicious, then the attack may not be noticed. For instance, if Bob opens a gym door, the door opens and lets Bob in without authentication. Instead it relays the messages back and forth to Bob's house and let its malicious owner gain access to Bob's house. If Bob does not notice on his Master Key's display that he is opening his house door, then the attack will occur successfully.

Mafia fraud attacks may not have countermeasures by cryptography alone. Presently, there are several representative solutions to counter the attacks without physically isolating claimants (devices). First, location information may be integrated into an authentication protocol as proposed in [80]. Second, measuring the transmission time between a claimant and a verifier, and then one can determine whether the distance between the two is within the expectation [81, 82]. Recent improvements based on location and time information may be found in [83]. Third, based on the assumption that an eavesdropper is not able to monitor all communication channels, a large number of channels are simultaneously used to obscure some real communication channels [84]. These approaches can be adapted and fit into our protocols. For instance, if the Master Key and a lock know their location information, then the code words can be also based on the location information. Thus, an attack will be easily detected from the location information. Moreover, the Master Key and a lock may measure their upper distance bound. Instead of sending a code word in the hash format in one message, the Master Key and a lock may send a bit at a time over multiple rounds and determine whether their distance is reasonable.

Securing the Master Key is critical. Losing it may be as serious as losing a key chain and/or a wallet. Finger recognition and tamper-resistant features may reduce the problem. Moreover, the Master Key may need multiple interfaces (physical contact, wired communications, and wireless communications) and may communicate over different radio frequencies to interact with various locks. These problems are important, but they are out of the scope of this paper.

The design of the Master Key assumes that all locks may be operated via the same kind of wireless link. It is likely that some locks may have different requirements and goals, and thus those locks might use different communication channels such as physical contact or different kinds of wireless links. If the Master Key operates many locks via a link, we may still use the approach as we discussed.

7.5. Summary

In this chapter, we propose the Master Key approach for entity authentication in pervasive computing environments. Our approach improves usability such that a person carries one device for various authentication purposes while it maintains the favorable properties of carrying multiple access tokens. The Master Key exchanges code words with locks securely and privately. Sensitive information, including identities and presence information, is protected from malicious outsiders via encryption and from malicious insiders via a probabilistic approach.

The current design of the Master Key does not support multiple groups of key owners. Thus, a lock will find up to one code word that matches in the Bloom filter format. However, if multiple groups are supported in a lock, the lock may find multiple code words that match several groups due to the false positive cases. The lock could determine the false positive cases by establishing multiple sessions and exchanging messages with the Master Key, but in the group key type this violates the privacy feature and thus the Master Key owner will not be sure that his shared secret for the lock is the same as other owners. We are designing an approach to support multiple groups, while the group key type still maintains its desirable privacy feature. In the meantime, we are designing an approach to make the revocation of a group key easier.

The design of exchanging a few bits to protect privacy as we discussed in Section 7.2.3 and 7.3.1 is conservative. A more aggressive approach could be let a lock and its key owner exchange a code word with more bits when $p(key\ owner)$ is small. However, the challenge is that a selfish lock that is only concerned about its overhead will sacrifice its key owners' privacy. We are designing an approach to properly balance the overhead and privacy properties.

CHAPTER 8. CONCLUSION AND FUTURE WORK

In this dissertation, we first presented comprehensive surveys about: service discovery protocols for pervasive computing environments, and authentication and entity authentication in pervasive computing environments. Then, we proposed two models for secure service discovery in public environments. The Splendor approach enables unfamiliar parties to provide and access network services based on PKI and location information. Whereas the two models discussed in Chapter 4 use hidden channels to facilitate secure ad hoc service discovery. Next, we discussed the PrudentExposure and progressive service discovery models. The former model automates private authentication among users and service providers. It discovers existing service providers in the vicinity and selects necessary credentials for authentication. The latter model improves personal privacy based on the PrudentExposure model. Personal privacy is protected and only exposed when exposure is necessary. Chapter 7 further extends the PrudentExposure and progressive models and applies the security solutions to entity authentication in pervasive computing environments.

The four security and privacy models (in Chapter 3, Chapter 4, Chapter 5, and Chapter 6) that we proposed do not solve all problems for secure and private service discovery in pervasive computing environments. Instead, this research opens up many research problems, which need to be addressed. Many more models may be proposed to address different service discovery situations, because it is unlikely that one model will be optimal for all situations. The key requirements for our future design will still be secure, private, and convenience.

Another research direction is to design a mechanism to compare models – the models that we proposed, others' models, and emerging models. For different situations, the mechanism automatically selects optimal models for users and service providers to interact with each other. The most challenging issue is how to quantitatively compare the approaches in terms of security, privacy, and convenience.

The models that discussed in Chapter 5 and Chapter 6 assume that users and service providers know each other and share secrets. In pervasive computing environments, a user and a service provider may not know each other. To enable such users and service providers discover and provide services is more difficult. One may use PKI as we use in Chapter 3 and Chapter 4. Nevertheless, such authentication approaches do not protect privacy because a user and a service provider have to expose their identities in the first place.

The Master Key design addresses entity authentication in pervasive computing environments. We are also interested in providing more general solutions for authentication in pervasive computing environments. For instance, when Bob and his son use the same remote control, Bob and his son should see different sets of channels. Proper authentication in this scenario is sophisticated, yet it provides great convenience to users.

BIBLIOGRAPHY

[1] T. Kindberg and A. Fox, "System Software for Ubiquitous Computing," *IEEE Pervasive Computing*, pp. 70-81, 2002.

[2] R. Anderson, *Security Engineering: A Guide to Building Dependable Distributed Systems*: John Wiley & Sons, 2001.

[3] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley, "The design and implementation of an intentional naming system," presented at 17th ACM Symposium on Operating Systems Principles (SOSP '99), Kiawah Island, SC, 1999.

[4] T. Hodes, S. Czerwinski, B. Zhao, A. Joseph, and R. Katz, "An Architecture for Secure Wide-Area Service Discovery," *ACM Wireless Networks Journal*, vol. 8, pp. 213-230, 2002.

[5] M. Nidd, "Service Discovery in DEAPspace," *IEEE Personal Communications*, pp. 39-45, 2001.

[6] Sun Microsystems, "Jini Technology Core Platform Specification," June 2003, http://wwws.sun.com/software/jini/specs/.

[7] UPnP.Forum, "Universal Plug and Play Device Architecture 1.0," UPnP Forum, Dec., 2003, http://www.upnp.org/resources/documents/CleanUPnPDA101-20031202s.pdf.

[8] S. Cheshire and M. Krochmal, "DNS-Based ServiceDiscovery," Apple Computer, Feb. 14, 2004, http://files.dns-sd.org/draft-cheshire-dnsext-dns-sd.txt.

[9] Salutation Consortium, "Salutation Architecture Specification," 1999, ftp://ftp.salutation.org/salute/sa20e1a21.ps.

[10] E. Guttman, C. Perkins, J. Veizades, and M. Day, "Service Location Protocol, Version 2," 1999, http://www.ietf.org/rfc/rfc2608.txt.

[11] Bluetooth SIG, "Specification of the Bluetooth System," Nov. 4, 2004, http://www.bluetooth.org/.

[12] W3C, "Web Services Architecture," Feb. 11, 2004, http://www.w3.org/TR/ws-arch/wsa.pdf.

[13] D. Bryan, V. Draluk, D. Ehnebuske, T. Glover, A. Hately, Y. L. Husband, A. Karp, K. Kibakura, C. Kurt, J. Lancelle, S. Lee, S. MacRoibeaird, A. T. Manes, B. McKee, J. Munter, T. Nordan, C. Reeves, D. Rogers, C. Tomlinson, C. Tosun, C. v. Riegen, and P. Yendluri, "UDDI Version 2.04 API Specification," 2002, http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.pdf.

[14] Bluetooth SIG, "Specification of the Bluetooth System -- Core," Version 1.1, 2001.

[15] M. Balazinska, H. Balakrishnan, and D. Karger, "INS/Twine: A Scalable Peer-to-Peer Architecture for Intentional Resource Discovery," presented at Pervasive 2002 - International Conference on Pervasive Computing, Zurich, Switzerland, 2002.

[16] Sun Microsystems, "Jini™Technology Core Platform Specification," Version 1.2, Sun Microsystem, 2001.

[17] B. Miller, "Mapping Salutation Architecture APIs to Bluetooth Service Discovery Layer," Version 1.0, Bluetooth SIG, 1999.

[18] S. Czerwinski, B. Y. Zhao, T. Hodes, A. Joseph, and R. Katz, "An Architecture for a Secure Service Discovery Service," presented at Fifth Annual International Conference on Mobile Computing and Networks (MobiCom '99), Seattle, WA, 1999.

[19] B. Bloom, "Space/Time Trade-offs in Hash Coding with Allowable Errors," *Communications of ACM*, pp. 422-426, 1970.

[20] B. A. Miller, T. Nixon, C. Tai, and M. D. Wood, "Home Networking with Universal Plug and Play," *IEEE Communications Magazine*, pp. 104-109, 2001.

[21] S. Cheshire, "Discovering Named Instances of Abstract Services using DNS," Apple Computer, 2002, http://files.dns-sd.org/draft-cheshire-dnsext-dns-sd.txt.

[22] Apple Computer Inc, "Rendezvous Web Site," 2003, http://developer.apple.com/macosx/rendezvous/.

[23] W3C, "Web Services Description Language (WSDL) Version 2.0," Aug. 3, 2004, http://www.w3.org/TR/2004/WD-wsdl20-20040803.

[24] T. Kindberg, J. Barton, J. Morgan, G. Becker, D. Caswell, P. Debaty, G. Gopal, M. Frid, V. Krishnan, H. Morris, J. Schettino, and B. Serra, "People, Places, Things: Web Presence for the Real World," 2000, http://cooltown.hp.com/dev/wpapers/index.asp.

[25] K. Edwards, M. Newman, and J. Sedivy, "The Case for Recombinant Computing," Palo Alto Research Center, Palo Alto Xerox Palo Alto Research Center Technical Report CSL-01-1, April 20 2001.

[26] Sun Microsystems, "Jini™Technology Surrogate Architecture Specification," Sun Microsystem, July 2001, http://surrogate.jini.org/sa.pdf.

[27] C. Ellison, "Home Network Security," *Intel Technology Journal*, vol. 06, pp. 37-48, 2002.

[28] Bluetooth SIG Security Expert Group, "Bluetooth Security White Paper," 2002, http://grouper.ieee.org/groups/1451/5/Comparison%20of%20PHY/Bluetooth_24Security_Paper.pdf.

[29] C. Ellison, "UPnP™ Security Ceremonies V1.0," Intel Co., Oct. 3 2003, http://www.upnp.org/download/standardizeddcps/UPnPSecurityCeremonies_1_0secure.pdf.

[30] F. Sommers, "Jini Starter Kit 2.0 tightens Jini's security framework," *Java World*, 2003.

[31] F. Zhu, M. Mutka, and L. Ni, "PrudentExposure: A Private and User-centric Service Discovery Protocol," presented at 2nd IEEE Annual Conference on Pervasive Computing and Communications, Orlando, Florida, 2004.

[32] F. Stajano and R. Anderson, "The Resurrecting Duckling: Security Issues for Ad-hoc Wireless Networks," presented at 7th International Workshop on Security protocols, Cambridge, UK, 1999.

[33] F. Stajano and R. Anderson, "The Resurrecting Duckling -- what next?," presented at 8th International Workshop on Security protocols, Cambridge, UK, 2000.

[34] D. Balfanz, D. K. Smetters, P. Stewart, and H. C. Wong, "Talking to Strangers: Authentication in Ad-Hoc Wireless Networks," presented at 9th Annual Network and Distributed System Security Symposium, San Diego, CA, 2002.

[35] R. Anderson, F. Bergadano, B. Crispo, J.-H. Lee, C. Manifavas, and R. Needham, "A New Family of Authentication Protocols," *Operating Systems Review*, 1998.

[36] I. Zakiuddin, S. Creese, B. Roscoe, and M. Goldsmith, "Authentication in Pervasive Computing: Position Paper," presented at 1st International Conference on Security in Pervasive Computing, Boppard, Germany, 2003.

[37] F. Stajano, *Security for Ubiquitous Computing*: John Wiley & Sons, LTD, 2002.

[38] M. Burnside, D. Clarke, T. Mills, S. Devadas, and R. Rivest, "Proxy-Based Security Protocols in Networked Mobile Devices," presented at 17th ACM Symposium on Applied Computing, Madrid, Spain, 2002.

[39] M. Langheinrich, "A Privacy Awareness System for Ubiquitous Computing Environments," presented at UbiComp 2002, GÖTEBORG, SWEDEN, 2002.

[40] K. Zhang and T. Kindberg, "An Authorization Infrastructure for Nomadic Computing," HP Labs, HPL-2001-228 20010921, 2001.

[41] M. Abadi and C. Fournet, "Private Authentication," *Theoretical Computer Science*, vol. September, pp. 427-476, 2004.

[42] S. G. Halliday, "Introduction to Magnetic Stripe & Other Card Technologies," presented at SCAN-TECH ASIA 97, Singapore, 1997.

[43] Association for Payment Clearing Services, "UK CARD FRAUD LOSSES REACH £504.8M," 2005, http://www.epaynews.com/downloads/APACS%20UK%20Card%20Fraud%20PR%208%20March%202005.pdf.

[44] J. Ferrari, R. Mackinnon, S. Poh, and L. Yatawara, *Smart Cards: A Case Study*: IBM Corporation, 1998.

[45] Smart Card Alliance, "Smart Card Implementation Profiles," http://www.smartcardalliance.org/industry_info/profiles.cfm.

[46] J. Pearson, "Securing the Pharmaceutical Supply Chain with RFID and Public-key infrastructure (PKI) Technologies," Texas Instruments, June 2005, http://www.ti.com/rfid/docs/manuals/whtPapers/wp-Securing_Pharma_Supply_Chain_w_RFID_and_PKI_final.pdf.

[47] AXCESS Inc web site, "Personnel Access Control," 2005, http://www.axcessinc.com/.

[48] A. Juels, R. Rivest, and M. Szydlo, "Selective Blocking of RFID Tags for Consumer Privacy," presented at 10th ACM Conference on Computer and Communications Security, Washington D.C., 2003.

[49] S. A. Weis, S. Sarma, R. Rivest, and D. Engels, "Security and Privacy Aspects of Low-Cost Radio Frequency Identification Systems," presented at 1st International Conference on Security in Pervasive Computing, Boppard, Germany, 2003.

[50] P. Agrawal, N. Bhargava, C. Chandrasekhar, A. Dahya, and J. D. Zamfirescu, "The MIT ID Card System: Analysis and Recommendations," 2004, http://swiss.csail.mit.edu/6.805/student-papers/fall04-papers/mit_id/.

[51] Dallas Semiconductor, "Requirements of Remote Keyless Entry (RKE) Systems," Nov. 11 2004, http://www.maxim-ic.com/appnotes.cfm/appnote_number/3395.

[52] G. Goebel, "Codes, Ciphers, & Code breaking," Jun. 1 2004, http://www.vectorsite.net/ttcode.html.

[53] iButton home page, 2005, http://www.maxim-ic.com/products/ibutton/.

[54] A. Harter and A. Hopper, "A Distributed Location System for the Active Office," *IEEE Network*, vol. 8, 1994.

[55] M. Corner and B. Noble, "Zero-Interaction Authentication," presented at Conference on Mobile Computing and Networking (MobiCom), Atlanta, Georgia, USA, 2002.

[56] Ensure Technologies, http://www.ensuretech.com/products/technology/technology.html#HowXyLocWorks.

[57] A. Beaufour and P. Bonnet, "Personal Servers as Digital Keys," presented at 2nd IEEE Annual Conference on Pervasive Computing and Communications, Orlando, Florida, 2004.

[58] A. Menezes, P. v. Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*: CRC Press, 1996.

[59] F. Zhu, M. Mutka, and L. Ni,"Service Discovery in Pervasive Computing Environments," IEEE Pervasive Computing, vol. 4, No. 4, pp. 81-90, 2005.

[60]W. Stallings, *Cryptography and Network Security: Principles and Practice*, 2nd ed: Prentice Hall, 1998.

[61] S. Lloyd, C. Adams, and S. Kent, *Understanding Public-Key Infrastructure: Concept, Standards, and deployment considerations*: New Riders, 1999.

[62] D. Cooper, "A Model of Certificate Revocation," presented at Fifteenth Annual Computer Security Applications Conference, 1999.

[63] Visa and MasterCard, "SET Secure Electronic Transaction Specification Book 2: Programmer's Guide," Version 1.0, 1997.

[64] RSA Security Inc, "Scalability Proof of Concept: RSA Keon Certificate Authority Eight Million Certificate Test," 2002.

[65] M. Burrows, M. Abadi, and R. Needham, "A Logic of Authentication," *ACM Transactions on Computer Systems*, 1990.

[66] E. Amoroso, *Fundamentals of Computer Security Technology*. New Jersey: PRT Prentice Hall, 1994.

[67] M. Winslett, T. Yu, K. E. Seamons, A. Hess, J. Jacobson, R. Jarvis, B. Smith, and L. Yu, "Negotiating Trust on the Web," *IEEE Internet Computing*, pp. 30-37, 2002.

[68] F. Zhu, M. Mutka, and L. Ni, "Splendor: A Secure, Private, and Location-aware Service Discovery Protocol Supporting Mobile Services," presented at 1st IEEE Annual Conference on Pervasive Computing and Communications, Fort Worth, Texas, 2003.

[69] R. Want, T. Pering, G. Danneels, M. Kumar, M. Sundar, and J. Light, "The Personal Server - Changing the Way We Think about Ubiquitous Computing," presented at 4th International Conference on Ubiquitous Computing, Goteborg, Sweden, 2002.

[70] iButton home page, http://www.maxim-ic.com/products/ibutton/.

[71] M. Bellare, R. Canettiy, and H. Krawczykz, "Keying Hash Functions for Message Authentication," presented at Advances in Cryptology–CRYPTO '96 (LNCS 1109), 1996.

[72] L. Fan, P. Cao, J. Almeida, and A. Broder, "Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol," *IEEE/ACM TRANSACTIONS ON NETWORKING*, vol. 8, pp. 281-293, 2000.

[73] S. Ross, *Introduction to Probability Models*, eighth ed: Academic Press, 2003.

[74] J. Rice, *Mathematical Statistics and Data Analysis*, Second ed: Duxbury Press, 1995.

[75] F. Zhu, M. Mutka, and L. Ni, "A Private, Secure and User-centric Information Exposure Model for Service Discovery Protocols," *IEEE Transactions on Mobile Computing*, vol. 5, pp. 418-429, 2006.

[76] Schlage, "History of Locks," http://professional.schlage.com/about_us_historyoflocks.asp.

[77] M. Blaze, "Rights Amplification in Master-Keyed Mechanical Locks," *IEEE SECURITY & PRIVACY*, vol. 1, pp. 24-32, 2003.

[78] F. Zhu, W. Zhu, M. Mutka, and L. Ni, "Expose or Not? A Progressive Exposure Approach for Service Discovery in Pervasive Computing Environments," presented at 3rd IEEE Annual Conference on Pervasive Computing and Communications, Kauai island, Hawaii, 2005.

[79] A. Broder and M. Mitzenmacher, "Network Applications of Bloom Filters: A Survey," *Internet Mathematics*, vol. 1, 2005.

[80] Y. Desmedt, "Major security problems with the 'unforgeable' (feige)-fiat-shamir proofs of identity and how to overcome them," presented at SecuriCom '88, Paris, France, 1988.

[81] T. Beth and Y. Desmedt, "Identification tokens -- or: Solving the chess grandmaster problem," presented at Advances in Cryptology Crypto '90, 1991.

[82] S. Brands and D. Chaum, "Distance-bounding protocols," presented at Advances in Cryptology – EUROCRYPT '93, Lofthus, Norway, 1993.

[83] Y.-C. Hu, A. Perrig, and D. Johnson, "Packet Leashes: A Defense against Wormhole Attacks in Wireless Networks," presented at 22nd Annual Joint Conference of IEEE Computer and Communication Societies (INFOCOM 2003), San Francisco, CA, 2003.

[84] A. Alkassar, C. Stüble, and A.-R. Sadeghi, "Secure object identification: or: solving the Chess Grandmaster Problem," presented at the 2003 workshop on New security paradigms, Ascona, Switzerland, 2003.