# Private and Secure Service Discovery via Progressive and Probabilistic Exposure

Feng Zhu, Wei Zhu, Matt W. Mutka, *Senior Member*, *IEEE*, and Lionel M. Ni, *Fellow*, *IEEE*

**Abstract**—The involvement of only the necessary users and service providers for service discovery in pervasive computing environments is challenging. Without prudence, users' and service providers' requests or service information, their identities, and their presence information may be sacrificed. We identify that the problem 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. In this paper, we propose a progressive and probabilistic approach to solve the problem. Users and service providers expose partial information in turn and avoid unnecessary exposure if there is any mismatch. Although 1 or 2 bits of information are exchanged in each message, we prove that the process converges and that the false-positive overhead decreases quickly. Experiments and hypothesis tests show that security properties hold. We implemented the approach and the performance measurements show that the approach runs efficiently on PDAs.

**Index Terms**—Authentication, pervasive computing, privacy, probabilistic, security.

## 1 Introduction

In pervasive computing environments, intelligent devices are ubiquitously embedded within our personal belongings, homes, offices, and even public environments. These devices provide us various network services (services for short). Via service discovery protocols, these services are discovered just in time. Client devices and services automatically configure themselves without users' involvement. Much research has been conducted on service discovery, as reviewed in [1]. However, the problem of involving only necessary service providers and users in a service discovery session has not been well addressed. If unnecessary users and service providers are involved, then security and privacy may be sacrificed. Services may be illegally discovered or accessed and personal privacy may be exposed and inferred.

For traditional network service accesses, it is not difficult to involve only necessary and legitimate service providers and users. Usually, a user explicitly specifies a service and supplies a credential such as a username and password pair to authenticate with a service provider. Then, the service provider verifies the user and checks the user's privilege. The user has prior knowledge of the service, service provider, credential, and relationship among them. Nevertheless, in pervasive computing environments, a user may not have such knowledge.

Challenges arise when environments change. First, a user may interact with many more services and service providers in pervasive computing environments than in conventional computing environments. For instance, a room may be saturated with hundreds of devices and services. Furthermore, everyone may become a service provider. For example, if Bob shares his MP3 player with Alice, then Bob becomes a service provider. A significant growth in the number of services and service providers makes it difficult to memorize the relationships among the services, service providers, and credentials. Second, pervasive computing environments are extremely dynamic. Devices and services may be unattended, services are added and removed, service providers' mobility causes the devices that they wear and carry to move, and partial failures cause services to be inaccessible. The dynamic property of pervasive computing hinders users to have complete knowledge of the relationship among services, service providers, and credentials.

Without such knowledge, the problem to involve only necessary service providers and users becomes difficult when users and service providers have privacy concerns. If a user is too cautious to interact with a service provider, then a user may miss the opportunity to access a service and a service provider misses an opportunity to serve a user. However, unnecessary interaction between a user and a service provider may expose a user's intent (what service a user is looking for), his credentials, and presence information. Similarly, a service provider may unnecessarily expose his service information, identity, and presence information.

Many service discovery protocols have been proposed, but it seems that no protocol addresses the problem without sacrificing security, privacy, or convenience. Several protocols and their security extensions adopt the traditional approach such that users start service discovery by supplying credentials together with service discovery requests [2], [3], [4], [5], [6], [7]. Although service providers attain security and privacy, users unavoidably experience inconvenience, unnecessary privacy exposure, and missed opportunities to access services. In the trusted central server

- *F. Zhu, W. Zhu, and M.W. Mutka are with the Department of Computer Science and Engineering, 3115 Engineering Building, Michigan State University, East Lansing, MI 48824. E-mail: {zhufeng, mutka, zhuwei}@cse.msu.edu.*
- *L.M. Ni is with the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong. E-mail: ni@cse.ust.hk.*

*Manuscript received 11 Mar. 2006; revised 22 Oct. 2006; accepted 18 Jan. 2007; published online 2 Feb. 2007.*
*Recommended for acceptance by C. Shahabi.*
*For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0057-0306.*
*Digital Object Identifier no. 10.1109/TPDS.2007.1075.*

| User \ Service provider | No | Yes |
|---|---|---|
| No | Case 1 | Case 3 |
| Yes | Case 2 | Case 4 |

Fig. 1. Four cases of privacy concerns.

approach [8], service providers announce their service information to a central server system and users discover services via the server system. The design is secure and only involves necessary users and service providers besides the server system. Nevertheless, both users and service providers expose their privacy to the central server system. In PrudentExposure [9], only users and service providers that share secrets discover and communicate with each other, but there is still a privacy leak among insiders. For example, if Bob only shares an MP3 player with Alice, then it is unnecessary to contact Bob when Alice discovers an electronic book.

We classify privacy concerns into four cases, as shown in Fig. 1. The four cases are the combinations resulting from 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, then 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.

In this paper, 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.

We modeled the progressive approach as a Markov chain. Mathematical analysis shows that our approach converges, although only a few bits of information are exchanged in each message. False-positive cases (unnecessary cases) happen due to the few bits of information exchange, but mathematical analysis further shows that our approach addresses those cases efficiently and the probabilities of false-positive matches are known in each state. Moreover, we generated billions of test cases and did hypothesis tests to verify that exchanging partial information has the desired security properties. Additionally, we implemented the approach. Performance measurements showed that our approach is efficient. It only introduces very limited overhead and it takes up to 100 ms for a user and a service provider to determine necessary exposure between them on PDAs.

The rest of the paper is structured as follows: In Section 2, we discuss related work. Then, we present a formal problem specification in Section 3. Section 4 illustrates our progressive exposure design. In Section 5, we justify our claims through analysis and experimentation. In Section 6, we outline our future work and conclude our contribution.

## 2 Related Work

In this section, we discuss some representative service discovery protocols that provide security and privacy features. Then, we discuss other related research. For a wider survey of service discovery protocols, refer to our survey paper published elsewhere [1].

Existing secure and private service discovery protocols may be classified into three categories:

*Traditional authentication and authorization solutions.* In this category, a user or a device is responsible for supplying correct credentials to discover and access services, for example, Universal Plug and Play (UPnP) Security [6], [7] and Bluetooth Security [5]. UPnP Security provides many authorization methods including access control lists, authorization servers, authorization certificates, and group definition certificates, and users are assumed to supply correct credentials. In the trusted discovery mode of Bluetooth Security, services only interact with a device that shares a common secret. In this category, service providers may easily protect their privacy. If a user does not have the privilege to discover and access a service, then a service provider can remain silent. Users usually have to expose their identities and service requests to service providers. Therefore, users may unnecessarily expose their sensitive information when service providers do not provide the requested services. Moreover, users need to memorize the relation among services, service providers, and credentials. Otherwise, they may not be able to supply correct credentials and, thus, they lose opportunities to access services. Since memorizing the relation causes very poor usability in pervasive computing environments, users may bypass security and privacy features.

*Trusted central servers.* In Secure Service Discovery Service (SSDS) [8], servers are in a hierarchical structure. A server at the leaf level controls services at a place, whereas a server at a higher level aggregates information on the lower level servers. A user uses his public key to authenticate with a local server and discover services. Service providers specify user privileges and register services with local servers. Furthermore, communication among the parties is encrypted. With the help from the servers, only proper and necessary users and service

providers will be involved in a service discovery session. The approach provides good usability because a user may only need one credential (his public key) to discover services. Using one credential, however, is also a disadvantage because a credential broken in an application will jeopardize other applications. Moreover, users and service providers expose their sensitive information to the central server system. The system knows where the service providers are and what they have. Similarly, the system knows where the users are and their intents. Unlike the trusted central server approach, we assume that each service provider manages his own services and decides whether to expose sensitive information.

*Automated service provider discovery and credential management.* In our previous work, PrudentExposure [9] users do not need to actively identify existing services and service providers. Instead, a user's program discovers service providers from whom he acquires credentials via code words. Specifically, code words are in the Bloom filter format [10]. Such a format enables users and service providers to identify each other in one round of message exchange, even if there are hundreds of service providers that a user may interact with. After identifying each other, a user and a service provider establish an encrypted channel to exchange requests and service information. The approach involves only users and service providers that share secrets, but a privacy leak still occurs when service providers do not provide requested services or users do not have privileges to access services. For example, Alice does not need to contact Bob when she is discovering electronic books if Bob does not share electronic books with her. In such cases, users' intents and presence information and service providers' identities and presence information are exposed, respectively. Although users and service providers share secrets, such exposure is not necessary. The progressive approach proposed in this paper distinguishes and avoids these unnecessary exposures during mutual partial exposures. Unlike the PrudentExposure approach, exposure in the progressive approach is based on both identities and service information.

Other work also influences our approach. Automated trust negotiation systems such as [11], [12], and [13] establish mutual trust between strangers on the Internet. Two parties, in turn, disclose part of their access control polices and submit required credentials until they reach mutual trust. For example, one party may require credit card information, whereas the other party may require seeing a certificate from a Good Business Bureau first. The systems protect not only the resources but also sensitive attributes and exposure policies during negotiation [14], [15]. A negotiation process will proceed, and further requirements will be exchanged when both parties meet the other party's requirement in the current round. Nevertheless, the systems cannot establish trust when both parties want the other party to expose certain information first (the chicken-and-egg problem). Our approach has a stronger assumption that a user and a service provider share a secret for service discovery. Thus, a user and a service provider understand encrypted partial identity and service information. In addition, automated trust negotiation systems focus on establishing mutual trust among strangers over the Internet, whereas we target users and service providers within a vicinity.

Expressing knowledge of a secret in a sequence of messages in Port Knocking [16] inspires our work. In order to connect to a service on a server, a client "knocks" on the server's firewall in a special sequence based on a shared secret between them. The knocking sequence is a serial of connection messages to different closed ports on the firewall. Another innovative approach is authentication on untrustworthy public Internet access points [17]. The authentication requires a user to correctly recognize a sequence of personal photos in a reasonable time. In our approach, a user and a service provider establish trust via a sequence of mutual authentication.

The challenge and response procedure in Zero-Knowledge proofs [18] and our approach are similar: authentication over multiple rounds and probabilistic. In Zero-Knowledge proofs, one specifies its identity and then answers a set of authentication questions that a verifier asks. Similar to public-key-based authentication, verifiers know how they can verify answers, but they do not know how they can generate answers. Zero-Knowledge proofs cannot be used to solve our problem. Because users and service providers must specify their identities initially, they sacrifice privacy. Otherwise, they do not know what authentication questions to ask. In addition, in our approach, a user and a service provider share a secret, but in Zero-Knowledge proofs, sharing a secret is unacceptable because their major goal is to avoid impersonation.

Much research addresses personal privacy when passive RFID tags are pervasively associated with personal belongings [19], [20], [21]. With limited gates available to security functionalities on RFID tags, light weighted cryptographic functions such as hashes are used to generate encrypted IDs. Only an RFID reader that shares a secret with a tag understands the encrypted ID or triggers a tag to send its encrypted ID. Our progressive approach not only prevents eavesdroppers to acquire privacy information but also avoids unnecessary exposure among users and service providers that share secrets.

## 3 Problem Definition

In this paper, we focus on the chicken-and-egg problem and use the following formal model to discuss and analyze our exposure approach. We assume that a user and a service provider share a secret before they interact with each other. We also assume that each service has a standard name:

- A user, $U$, shares unique secrets with a set of service providers, $\{S_k\}_{k\in N}$. A service provider, $S$, may have a set of users, $\{U_l\}_{l\in N}$.
- When a user wants to discover a service, a set of service providers, $\{S_m\}_{m\geq 0}$, is in the vicinity. For a service provider, $S_a \in \{S_m\}$, it is possible that $S_a \notin \{S_k\}$. In addition, it is possible that, at different times, $\{S'_m\} \neq \{S_m\}$ because service providers may move around.

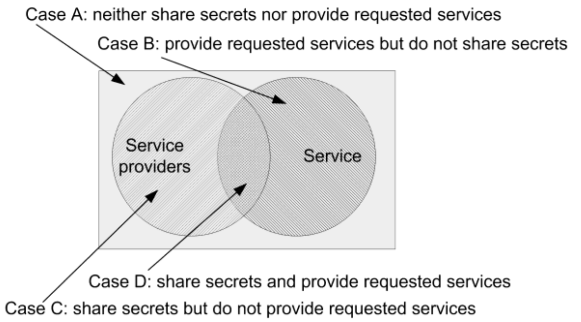


Fig. 2. Design goal of the progressive approach (find Case D).

- A service provider in the vicinity has a subset of services, $\{V_n\}_{n \in N}$, which matches a user's request at the discovery moment. The set $\{V_n\}$ may change at any time because of the dynamic property in pervasive computing environments. For example, new services are added and granted access to a user or partial failures cause services to be inaccessible.
- Service providers and users have privacy concerns. They want to protect their identities, existence information, service information, and service requests. 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. Note that, even if a user and a service provider share a secret, they may not be legitimate and exposure is not necessary. This situation is not addressed in existing solutions.

The constraints are that users and service providers want to interact with each other, whereas their privacy is protected. Fig. 2 shows our design goal from a user's perspective: Find Case D, in which a service provider provides the service that a user is looking for, and they share a secret. That is, for some service providers, $S \subset \{S_k\} \cap \{S_m\} \neq \phi$, and for those $S$, $\{Vn\} \neq \phi$ (from a service provider's perspective, the diagram is similar). In addition, when the numbers of elements in $\{S_k\}$, $\{U_l\}$, and $\{V_n\}$ are large, Case D still needs to be identified efficiently.

## 4 A Progressive Exposure Approach

To solve the chicken-and-egg problem (find Case D), users and service providers expose partial (several bits of) encrypted information in turn, as shown in Fig. 3. 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, then the communication stops. If matches repeatedly occur, then at some point, the service provider and the user believe that the other party is legitimate with high probability. Finally, the two parties establish a connection for service usage.

During each round, the progressive protocol identifies and excludes unnecessary exposures (Cases A, B, and C). For those cases, sensitive information is preserved because

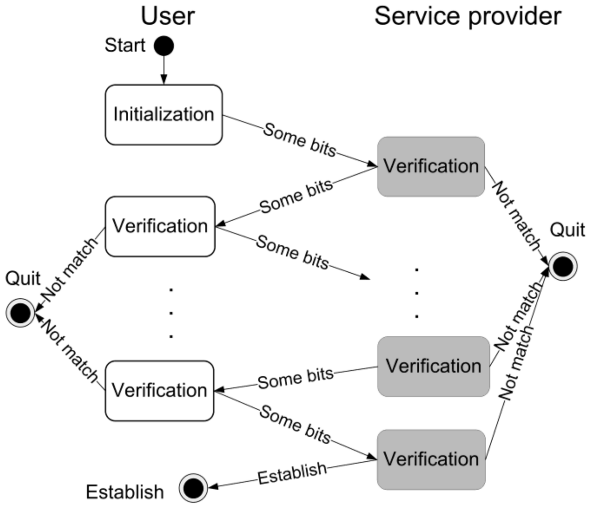


Fig. 3. Users and service providers expose partial identities and service information in turn.

only a few bits of sensitive information are exposed. A service provider usually provides a small set of services in comparison to all different types of services. Suppose that a unique ID represents each type of service and all IDs have the same length. On the average, each bit of a service ID excludes half of the types of services provided by a service provider. Thus, an unnecessary exposure is identified in several bits of service information exchange. However, to learn the requested service and available services, 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, neither users nor service providers can discern Case A, Case B, or Case C in Fig. 2. 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. Thus, in this case, the mismatch looks like Case B, but it is Case A.

In the remainder of this section, we first discuss how we can exchange encrypted sensitive information, so users and service providers that do not share secrets do not even acquire partial sensitive information. Then, we present the message formats and the 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.

### 4.1 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 this. If the service provider does not recognize the code word, he keeps silent.

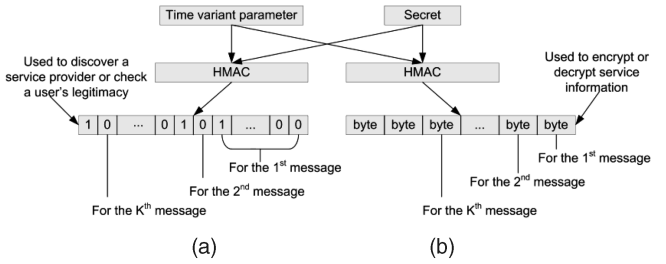


Fig. 4. (a) Generating a one-time code word to identify a user and a service provider. (b) Generating a one-time secret to protect service information.

Otherwise, he says another several bits of the code word and the user checks this. Via the interaction over multiple rounds, a user identifies existing service providers within the vicinity (for all $S$, $S \subset \{S_k\} \cap \{S_m\}$), and a service provider identifies a user $(U_i)$. Eavesdroppers, however, 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).

Fig. 4a 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 (HMACs), as proposed in [22]:

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

Therefore, a code word is the hash result.

The nature of the HMAC ensures that it is computationally difficult to find the shared secret from the hash results [18]. Thus, only a user and a service provider who share a secret can correctly generate and verify the code words. For any service provider, $S_a \notin \{S_k\}$, the user does not know who the service provider is, even if false-positive matches happen. Similarly, for any user, $U_b \notin \{U_l\}$, a service provider does not know who the user is.

### 4.2 Exchange Encrypted Service Requests and Available Services

Services are identified by unique IDs with the same length. Users and service providers use one-time secrets to protect service requests and service information, respectively. The generation of a one-time secret is shown in Fig. 4b. 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 [18]. According to [18], if the bytes (generated one-time secrets) that we use to encrypt service information are random, then our encoding method is computationally secure. We prove that the bytes are random and follow the uniform distribution over the integer set in Section 5.4.

TABLE 1
The Encoding Scheme for Services

| 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)

Users and service providers exchange service requests and service information in the encrypted form. Thus, users and service providers who share secrets understand service requests and available services. Among those users and service providers, only a subset of users and service providers whose requests match available services will find the subset $\{V_n\}$.

At a service provider's side, since partial information is exchanged, more than one type of service with the same initial bits may match a user's request. For example, if a user requests a service 10001, a service provider has two services: 10001 and 10101. If a user says the first 2 bits, 10, then the service provider will find two services starting with 10. To inform the user that there is more than one service that matches his request, we encode the possible combinations of 1 and 2 bits, as shown in Tables 1a and 1b, respectively. In this example, if the service provider replies with 1 bit of service information, then he will send the coding 10. If the service provider replies with 2 bits of service information, then he will send the coding 0101.

### 4.3 The Message Formats

In a discovery message, users and service providers exchange several bits of code words and service information. Only if the bits of a code word and service information match would one provide more bits.

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. Fig. 5a shows the message format. 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 Fig. 5b (the number of bits in the messages will be discussed in detail in Section 4.5).

The first message is sent as a broadcast message or a User Datagram Protocol (UDP) multicast message for the minimum configuration overhead. The following messages between a user and a service provider are sent via Transmission Control Protocol (TCP) unicast to guarantee

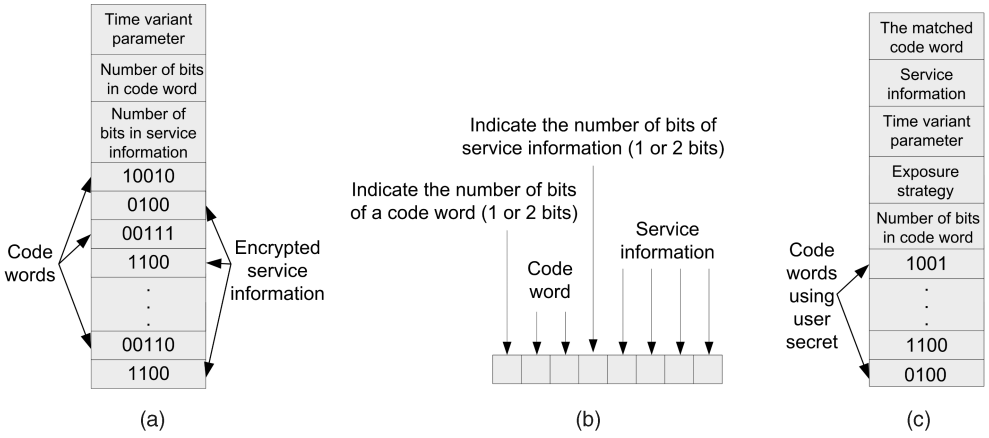


Fig. 5. Message formats. (a) First message. (b) Following messages. (c) Alternative second message.

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.

$CB_{USi}$ is bits of a code word generated from the domain secret shared between U and the $i^{th}$ S.

$CW_{SUl}$ is bits of a code word generated from the user secret shared between $l^{th}$ U and S.

$K_{USi}$ is a secret generated from the domain secret that U and $i^{th}$ S use to encrypt and decrypt messages.

$K_{SUl}$ is a secret generated from the user secret that $l^{th}$ U and S use to encrypt and decrypt messages.

SRB is bits of the requested service information.

SAB is bits of the available service information.

$Y^j$ is the element Y that is sent in the $j^{th}$ message.

ES is an exposure strategy.

$\{\}^n$ is a set of n elements, where $n \in N$.

Q is a message indicates the communication stops.

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

| Message number | Sender/Receiver | Message |
|---|---|---|
| 1 | U→S: | $R_U$, $T_U$, $\{CB^1_{USi}, (SRB^1)_{KUSi}\}^n$ |
| 2 | S→U: | $R_U$, $T_U$, $R_S$, $T_S$, $CB^1_{USi}$, $(SAB^2)_{KUSi}$, $\{CW^2_{SUl}\}^m$, ES |
| 3 | U→S: | $R_S$, $T_S$, $CW^2_{SUl}$, $CW^3_{SUl}$, $(SRB^3)_{KSUl}$ |
| 4 | S→U: | $R_S$, $T_S$, $CW^4_{SUl}$, $(SAB^4)_{KSUl}$ |
| A | U→S: or S→U: | $R_S$, $T_S$, Q |
| B | S→U: | $R_S$, $T_S$, P |

Fig. 6. The security protocol using domain secrets and user secrets.

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 several bits are 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. Fig. 5c 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 Fig. 5b. To revoke a user's service discovery privilege, the user secret is invalidated by the service provider, whereas updating the domain secret may not be imminent.

### 4.4 The Detailed Security Protocols

Fig. 6 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.

### 4.5 Predictable Exposure

In this section, we discuss the exposure from a service provider's point of view. From a user's point of view, the

TABLE 2
Number of Bits to Expose in a Code Word versus the Number of Credentials that a User Has

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

exposure is similar. (We use "*not user*" and "*user*" to denote $U_b \notin \{U_l\}$ and $U_c \in \{U_l\}$, respectively.) When verifying code word bits, a service provider finds either a match or a mismatch. If a mismatch occurs, then he knows that the other party does not know the shared secret. If a match happens, then he does not know whether the match is a false-positive match because only part of a code word is compared. Therefore, given that 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 that 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

$$p(match \mid not\ user) = 1 - \left(1 - \frac{1}{2^{number\ of\ bits}}\right)^{number\ of\ code\ words}. \quad (1)$$

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 percent. A user may simply select the number of bits to expose according to Table 2 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. Afterward, the false-positive rate will decrease by half for each additional bit exchanged.

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

From the preceding two probabilities, we have

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

Fig. 7 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 or a user does not have privilege). It depends on three facts: the probability that the service provider has the service $p(service)$, the probability of false-positive matches $p(match \mid not\ service)$, and the number of services that the service provider has. A user may not know how many services are provided by a service provider since available services may change from time to time. Thus, the user, by default, sends 4 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 the number of bits exchanged after the first message for a service provider with 320 services, as shown in Fig. 8. If a service provider has 160 (or 640) services, then he needs to exchange one less (or more) message to reach the same probability.

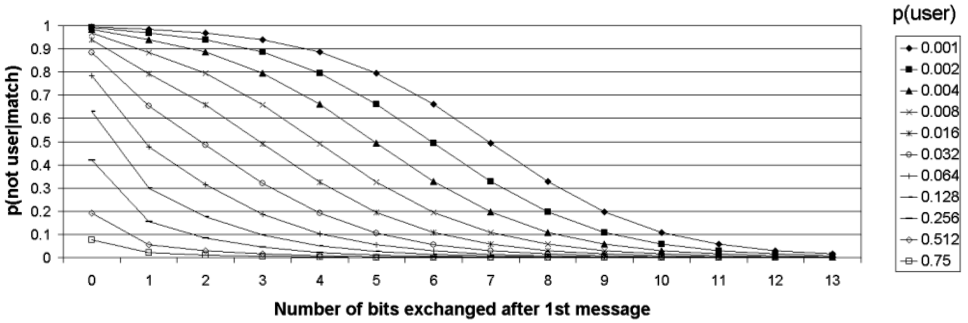


Fig. 7. $p(not\ user \mid match)$ decreases as the number of code word bits exposure increases after the first message.

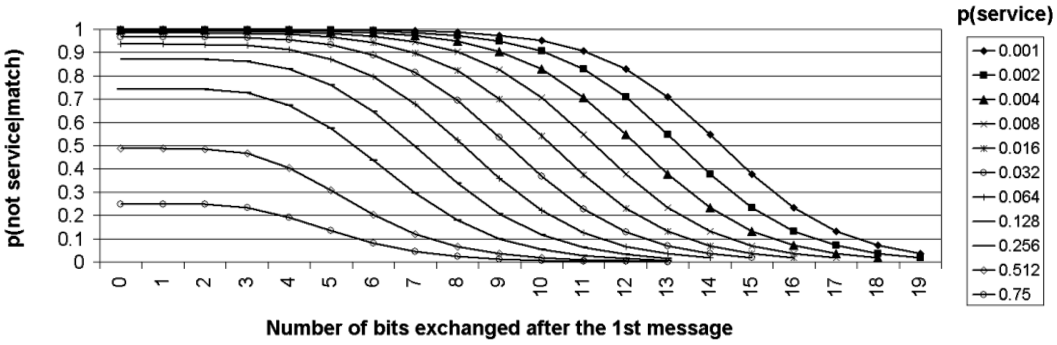


Fig. 8. $p(not\ service \mid match)$ decreases as the number of service information bits exchanged increases after the first message (the graph shows a service provider with 320 services).

TABLE 3
Number of Bits to Exchange to Reach a Critical Value (Less Than 5 Percent) 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) / Service | 0.001 | 0.002 | 0.004 | 0.008 | 0.016 | 0.032 | 0.064 | 0.128 | 0.256 | 0.512 | 0.75 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 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)

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

### 4.6 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 percent for both probabilities. If the probabilities are less than the critical values during a discovery process, then the service provider considers that the user's legitimacy is high enough and therefore 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. Tables 3a and 3b 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 services, $p(user)$ 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 discussed in Section 4.5.

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, that is, 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 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 that one message only takes about 4 ms 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, then 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.

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:

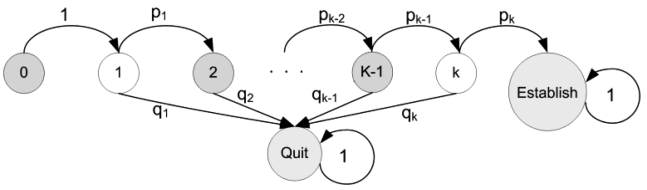


Fig. 9. Message exchange process expressed as a Markov chain.

First, if a service provider does not know the shared secret, then 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, then exchanging messages more than necessary does not offer him any better payoff. Conversely, if the service provider exchanges messages less than necessary, then 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, then 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. If a service provider cheats, then it can be easily detected via wild card search, which is available in many service discovery protocols.

## 5 System Evaluation

In this section, we first discuss three mathematical properties of our approach, namely, the false-positive overhead, code word conflicts, and convergence. Next, we present hypothesis tests to verify that security properties hold. Last, we measure the performance of our protocol.

### 5.1 False-Positive Overhead Decreases Exponentially

To analyze the false-positive overhead, we redraw the interaction between a user and a service provider (Fig. 3) as a Markov chain, as illustrated in Fig. 9. States "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, then the process goes to the next state (exchange more bits). Otherwise, the process goes to the absorbing state "*Quit*." If both parties are legitimate, then they always reach the "*Establish*" state. If at least one party is not legitimate (the exposure is unnecessary), then the process should go to the "*Quit*" state. Since there are false-positive matches, the process may go to the next state.

Supposing that an exposure is unnecessary, the probability that the process goes to the next state is $p_i$ and the probability that the process goes to the "*Quit*" state is $q_i$. We calculate $p_i$ from

$$p_i = \frac{p(match \cap not\ user) \times p(match \cap not\ service) + p(match \cap user) \times p(match \cap not\ service)}{p(not\ user) + p(user \cap not\ service)}.$$

Based on the calculation of the mean time spent in transient states [23], we calculate the probabilities that the Markov chain makes a transition into state "$i$," given that it starts from state "0":

$$S = (I - P_T)^{-1},\ where\ S\ is\ a\ matrix\ of\ values\ of\ s_{i,j}, the\ time\ periods\ in\ state\ j,\ given\ that\ it\ starts\ in\ i,$$

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

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

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, then 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 4, false-positive rates decrease exponentially.

### 5.2 Code Word Conflicts

In Section 4.5, we suggest that a user specifies 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 bits of the code words are sent in the first message, then there are about 4.5 code words that have conflicts with other code words. Therefore, when generating code words, if a user finds that the first several bits of a code word are already used for a service provider, then 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

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

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

**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 $E(codeword\ conflicts) = m - E(X_m)$:

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

TABLE 4
Probabilities that the Process Goes to the Next State, Given that an Exposure Is Not Necessary

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

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. Thus,

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

Therefore,

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

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

$$E(codeword\ conflicts) = m - \left(1 - \left(1 - \frac{1}{n}\right)^m\right) \times n.$$

□

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.

### 5.3 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 5.2), 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 Fig. 9. If there is a mismatch, then one party quits. If both the code word and the service information match, then 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 [23]. 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. □

### 5.4 Hash Results Follow the Uniform Distribution over the Integer Set

In Section 4, we assumed 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, then our encryption method for service information may not be computationally secure. Moreover, if the assumption does not hold, then the probabilities $p(match \mid not\ user)$, $p(not\ user \mid match)$, 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 the 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 [24]. 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 time stamp and 14 1-byte random numbers are used as a TVP. Next, we use the mechanism discussed in Section 4.1 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 percent as the significance level. The test results do not show evidence to reject the two hypotheses (the detail tests and results are presented in the Appendix). Therefore, we believe that the hash results follow the uniform distribution over an integer set.

### 5.5 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, 64-Mbyte 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 2 megabits per second (Mbps). Our software is developed using Microsoft eMbedded Visual C++ 3.0 and running on Microsoft PocketPC 3.0.

The experimental results show that our protocol is efficient on the PDAs. Table 5 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 discussed in Section 5.2, is used. The discovery process between a user and a service provider takes up to 100 ms. Each additional service provider involved takes another 30 ms. Therefore, within a reasonable time (a few seconds), a user can finish the discovery process.

TABLE 5
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 |

## 6 Conclusion and Future Work

In this paper, 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 hypothesis tests, we demonstrate that the security properties hold. Performance measurements show that our protocol runs efficiently in most cases on devices such as PDAs.

The progressive exposure approach might be used in applications besides service discovery. In general, if two parties communicate with each other and have privacy concerns, then they might have the chicken-and-egg problem. That is, both parties want the other party to expose some information first before proceeding. When the problem happens, both parties might progressively expose partial information to reach mutual trust.

Our approach has its limitation in extreme cases. The progressive is not efficient when many users and service providers are present in a place, for example, a stadium. A discovery process will cause many false-positive matches, especially in the first reply messages, as discussed in Section 5.1. Suppose a user's request reaches 500 service providers who have the same number of services, $p(user)$, and $p(service)$ values, as discussed in the example in Section 5.1. Then, the first message on the average causes 307 replies. In addition to network problems (such as collisions), most interactions between the user and service providers may be wasted due to false-positive matches. 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 a precise approach and when using a progressive approach could be complex. Another possible solution is to reduce the transmission power and restart service discovery when collisions occur. With smaller discovery ranges, however, users may miss opportunities to access services that are not in ranges. 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 that 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, whereas mobile service providers who do not move around often 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 services in the hash form.

We are exploring other exposure strategies and analyzing the strategies in terms of privacy risk and benefit, efficiency, and security. Under the current model, exposures are measured in probability and we 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 versus exposing an MP3 player that a person carries. However, quantifying the privacy risk may be difficult.

## Appendix

For the first hypothesis, 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. Table 6a shows the test results. Only one test $(n = 18)$ is significant. However, it may be given a false result. Because, given that the significance level is 5 percent, it seems reasonable that 1 out of 17 experiments is false. Then, we do 20 experiments to test as $n$ equals 18. Table 6b shows that only 1 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 7a. 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 7b. Therefore, we do not reject the null hypothesis.

TABLE 6
Test Results for Code Words

| Number of bytes | Number of code words | Possible outcomes | df | $\chi^2$ | 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 | $\chi^2$ | 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)

*(a) Chi-square tests on different lengths of code words. (b) Revisit the 18th bit of code words. Twenty groups of tests with 409,600,000 code words each.*

TABLE 7
Test Results for One-Time Secrets

| Byte Number | $\chi^2$ | 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 | $\chi^2$ | 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)

*(a) Chi-square tests on the 20 bytes of one-time secrets. (b) Twenty groups of tests on byte 6.*

## ACKNOWLEDGMENTS

The authors are grateful to Dr. Lyudmila Sakhanenko and Dr. James Stapleton for their fruitful discussions. They thank the anonymous reviewers for their valuable comments that greatly helped them improve this paper. This paper is an extension of a paper that appeared in the *Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications* (March 2005).

## REFERENCES

[1] F. Zhu, M. Mutka, and L. Ni, "Service Discovery in Pervasive Computing Environments," *IEEE Pervasive Computing*, vol. 4, pp. 81-90, 2005.

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

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

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

[5] "Bluetooth Security," white paper, Bluetooth SIG Security Expert Group, http://grouper.ieee.org/groups/1451/5/Comparison%20of%20PHY/Bluetooth_24Security_Paper.pdf, 2002.
[6] C. Ellison, *UPnP Security Ceremonies V1.0,* Intel Co., http://www.upnp.org/download/standardizeddcps/UPnPSecurityCeremonies_1_0secure.pdf, Oct. 2003.
[7] C. Ellison, "Home Network Security," *Intel Technology J.,* vol. 6, pp. 37-48, 2002.
[8] S. Czerwinski, B.Y. Zhao, T. Hodes, A. Joseph, and R. Katz, "An Architecture for a Secure Service Discovery Service," *Proc. MobiCom,* 1999.
[9] F. Zhu, M. Mutka, and L. Ni, "A Private, Secure and User-Centric Information Exposure Model for Service Discovery Protocols," *IEEE Trans. Mobile Computing,* vol. 5, pp. 418-429, 2006.
[10] B. Bloom, "Space/Time Trade-Offs in Hash Coding with Allowable Errors," *Comm. ACM,* vol. 13, pp. 422-426, 1970.
[11] P. Bonatti and P. Samarati, "Regulating Service Access and Information Release on the Web," *Proc. Seventh ACM Conf. Computer and Comm. Security (CCS '00),* 2000.
[12] T. Yu and M. Winslett, "A Unified Scheme for Resource Protection in Automated Trust Negotiation," *Proc. IEEE Symp. Security and Privacy,* 2003.
[13] 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.
[14] W.H. Winsborough and N. Li, "Towards Practical Automated Trust Negotiation," *Proc. Third Int'l Workshop Policies for Distributed Systems and Networks (POLICY '02),* 2002.
[15] W.H. Winsborough and N. Li, "Protecting Sensitive Attributes in Automated Trust Negotiation," *Proc. ACM Workshop Privacy in the Electronic Soc. (WPES '02),* 2002.
[16] M. Krzywinski, "Port Knocking: Network Authentication across Closed Ports," *SysAdmin Magazine,* vol. 12, pp. 12-17, 2003.
[17] T. Pering, M. Sundar, J. Light, and R. Want, "Photographic Authentication through Untrusted Terminals," *IEEE Pervasive Computing,* pp. 30-36, 2003.
[18] A. Menezes, P.V. Oorschot, and S.A. Vanstone, *Handbook of Applied Cryptography.* CRC Press, 1996.
[19] Y. Nohara, S. Inoue, K. Baba, and H. Yasuura, "Quantitative Evaluation of Unlinkable ID Matching Schemes," *Proc. ACM Workshop Privacy in the Electronic Soc. (WPES '05),* 2005.
[20] M. Ohkubo, K. Suzuki, and S. Kinoshita, "Cryptographic Approach to 'Privacy-Friendly' Tags," *RFID Privacy Workshop, Massachusetts Inst. of Technology,* 2003.
[21] S.A. Weis, S. Sarma, R. Rivest, and D. Engels, "Security and Privacy Aspects of Low-Cost Radio Frequency Identification Systems," *Proc. First Int'l Conf. Security in Pervasive Computing (SPC '03),* 2003.
[22] M. Bellare, R. Canettiy, and H. Krawczykz, "Keying Hash Functions for Message Authentication," *Proc. 16th Ann. Int'l Conf. Advances in Cryptology (Crypto '96),* 1996.
[23] S. Ross, *Introduction to Probability Models,* eighth ed. Academic Press, 2003.
[24] J. Rice, *Math. Statistics and Data Analysis,* second ed. Duxbury Press, 1995.

**Feng Zhu** received the BS degree in computer science from East China Normal University, the MS degree in computer science and engineering from Michigan State University, the MS degree in statistics from Michigan State University, and the PhD degree from Michigan State University. He was a software engineer at Intel and is currently a program manager at Microsoft. His current research interests include pervasive computing, security for pervasive computing, computer networks, and distributed systems.

**Wei Zhu** received the BS degree in computer science from East China Normal University in 1994, the MS degree in computer science and engineering from Michigan State University in 2001, the MS degree in statistics from Michigan State University in 2004, and the PhD degree in computer science and engineering from Michigan State University in 2006. Her research interests include human-computer interaction, computer graphics, augmented reality, and multimedia systems. She is currently a software design engineer at Microsoft Corp.

**Matt W. Mutka** received the BS degree in electrical engineering from the University of Missouri, Rolla, the MS degree in electrical engineering from Stanford University, and the PhD degree in computer science from the University of Wisconsin, Madison. He is currently a professor in the Department of Computer Science and Engineering, Michigan State University, East Lansing, Michigan. He has been a visiting scholar at the University of Helsinki and a member of the technical staff at Bell Laboratories in Denver. His current research interests include mobile computing, sensor networking, pervasive computing, and multimedia networking. He is a senior member of the IEEE.

**Lionel M. Ni** received the PhD degree in electrical and computer engineering from Purdue University, West Lafayette, Indiana, in 1980. He is a chair professor at the Hong Kong University of Science and Technology (HKUST) and heads the Department of Computer Science and Engineering. He is also the director of the HKUST China Ministry of Education/Microsoft Research Asia IT Key Laboratory and the director of the HKUST Digital Life Research Center. He has chaired many professional conferences and has received a number of awards for authoring outstanding papers. He is a coauthor of the book *Interconnection Networks: An Engineering Approach* with Jose Duato and Sudhakar Yalamanchili (Morgan Kaufmann, 2002), *Smart Phone and Next Generation Mobile Computing* with Pei Zheng (Morgan Kaufmann, 2006), and *Professional Smartphone Programming* with Baijian Yang and Pei Zheng (Wrox Publishing, 2007). He is a fellow of the IEEE.