Expose or Not? A Progressive Exposure Approach for Service Discovery in Pervasive Computing Environments*

Feng Zhu[1] Wei Zhu[1] Matt W. Mutka[1] Lionel Ni[2]

[1]*Dept. of Computer Science and Engr*
Michigan State University
East Lansing, Michigan, USA
{zhufeng, zhuwei, mutka}@cse.msu.edu

[2]*Dept. of Computer Science*
Hong Kong University of Science and Technology
Kowloon, Hong Kong, China
ni@cs.ust.hk

Abstract

In pervasive computing environments, service discovery facilitates users to access network services by automating tedious manual configurations. When network services becomes pervasive, the number of service providers also increase dramatically. Because of security and privacy concerns, network services are segmented by service providers. Existing service discovery protocols, however, do not address how to facilitate users to properly identify and authenticate with existing service providers. Without prudence, sensitive information may be exposed. Conversely, with prudence both users and service providers prefer the other party to expose sensitive information first. We identify that even among legitimate users and service providers, there are privacy concerns that may be expressed as a chicken-and-egg problem. In this paper, we propose a progressive approach to solve the problem. Users and service providers expose minimal sensitive information in turn and identify necessary exposure during the process. Theoretical analysis, simulation, and experiments show that our approach protects sensitive information with little overhead.

1. Introduction

In traditional secure network service accesses, a user explicitly specifies a service's network address and supplies a credential (a user name and password pair or a certificate) to authenticate with a service provider. The user has *a priori* knowledge of the service, the service provider, the credential, and the relation among them. Imagine within pervasive computing environments, in which network services (services for short) become ubiquitous and embedded within our personal belongings, homes, and offices. Every person may become a service provider and a user. Both the number of services and service providers with which a user interacts dramatically increases. As a consequence, two new challenges emerge. First, as the number of services increases, manual efforts to configure devices for potential communications and maintain availability of services become overwhelming. Second, as the number of service providers increases, memorizing the relation between services, service providers, and credentials becomes burdensome. Service discovery as an essential element for service access and sharing in pervasive computing has been widely accepted [1]. Most existing service discovery protocols provide elegant solutions such as soft state and lease-based mechanisms and just-in-time driver installation to meet the first challenge [2]. Instead of tedious manual configuration, users designate services by names and attributes, and then protocols discover services and configure devices. Nevertheless, the second challenge is not well addressed.

Current service discovery solutions may be roughly classified into four approaches. First, insecure service discovery protocols allow anyone to discover and use anyone else's services [3-6]. Second, approaches may apply traditional access control solutions to secure services within each service provider [7-11]. Third, trusted servers may manage authentication and authorization centrally [12]. Fourth, a protocol may discover existing service providers at a moment, and the software that manages a user's credentials for associated service providers automatically authenticates with the service providers [13]. The first approach obviously sacrifices security and privacy. With the second approach, a user has to memorize services and their associated credentials. In addition, since services and service providers may be mobile and partial failures may happen to services, the user has to identify the existence of service providers

* This research is supported in part by NSF Grant No. 0334035, NIH Grant No. EB002238-01, Hong Kong RGC Grants DAG02/03.EG02, and AoE/E-01/99.

and/or the services. The third approach improves usability such that a user may only need one credential for service discovery. Nevertheless, from a service provider's perspective, he has to expose services to central servers and trust servers to manage services securely. From a user's perspective, he has to expose every service requests to central servers. The fourth approach seems to solve the second challenge since it properly identifies the legitimacy of users and service providers. However, there are two privacy issues: both a user and a service provider expose their presence information; a user exposes a service request to all recognized service providers. Both exposures may not be necessary. For example, Bob and his colleagues may provide each other privileges to access MP3 players, electronic books, digital pictures, etc. When Bob discovers services using the fourth approach, he discovers existing service providers, and then he authenticates and queries them for services. However, if the service is only offered by his office, it is not necessary for Bob to tell colleagues his service request. Moreover, although the involved user and service providers are all legitimate, Bob and his colleagues' presence information can be inferred. Thus, we identify that even among legitimate users and service providers there are privacy concerns that do not exist in traditional network service access.

Designing a service discovery protocol that protects sensitive information for both users and service providers is challenging. From users' point of view, it is prudent to authenticate and expose service requests only to necessary service providers. However, identifying the necessary service providers requires knowledge of the current existing services and service providers and their relations. Ideally, if service providers expose their existence and service information first, users can choose only necessary service providers to contact. Nevertheless, from the service providers' point of view, it is not prudent to reply when a request is from an illegitimate user or the requested service is not offered. The act of hiding by not responding not only saves computation power and energy, but also protects the presence information of a service provider. Ideally, if users expose their credentials and service requests first, a service provider can easily make a decision. Therefore, both users and service providers prefer that the other party exposes information first. The conflict between a service provider and a user becomes the chicken-and-egg problem.

In this paper, we propose a progressive approach to solve the conflict between users and service providers. Users expose partial information about who they are and what services they are seeking. Then, if service providers find matches (recognize the users and offer the requested services), they expose partial information about who they are and what services are available. Users and service providers in turn expose until they reach a certain confidence level to authenticate for service access. If there is any mismatch about the service or user information during the processes, the communication stops. We target environments in which users discover services within their vicinity via wired or wireless networks.

Compared to the fourth approach, the progressive approach not only properly identifies the legitimacy of users and service providers, but also differentiates to which legitimate service provider that a user should expose his sensitive information and determines whether a service provider should expose its sensitive information to a legitimate user. Our approach addresses these new difficult problems:

Privacy: Communication in each round is based on mutual matches. When mismatches are found, communications stop and only partial information is exposed, such that the other party receives uncertain sensitive information.

Security: When a small amount of information is exposed, the number of false positive matches between foreign parties increases. Illegitimate users or service providers may be involved in the communications. However, our approach secures sensitive information. Therefore, illegitimate parties do not understand the sensitive information.

Fairness: During the exposure process, neither users nor service providers may acquire additional sensitive information while exposing less than a very limited amount of sensitive information.

Adaptive: Our approach is adaptive to support users with different numbers of credentials and service providers with different numbers of services in different environments. The approach requires little processing and storage space.

We prove the mathematical properties of our progressive exposure approach. The exposure in terms of probability is known in each step. Our experiments show that our approach is efficient. It introduces very limited overhead even on mobile devices such as PDAs. We also do simulations to test hypotheses in our model.

The rest of the paper is structured as follows. In Section 2, we discuss related work. Section 3 presents our progressive exposure approach. In Section 4, we demonstrate our claims through analysis and experimentation. In Section 5, we outline our future work and conclude our contribution.

2. Related Work

A detailed comparison of service discovery protocols may be found in [2], and thus we omit discussion of insecure service discovery approaches. We discuss representative protocols for the other three approaches.

Applying traditional access control solutions. In the trusted discovery mode of Bluetooth Security [11], service information is only exposed to a device that shares a common secret with the service. The solution is appropriate for Bluetooth devices that have limited resources. However, if the solution is widely used in pervasive computing environments, a user needs to maintain many credentials because for each device a different credential may be used. Moreover, identifying the existence of a device is inevitable. Universal Plug and Play (UPnP) Security provides many authorization methods including access control lists, authorization servers, authorization certificates, and group definition certificates [7]. Our approach complements UPnP Security and provides a mechanism to properly and automatically supply credentials and expose existence information.

Trusted central servers. In Secure Service Discovery Service (SSDS) [12], servers are in a hierarchical structure. A server at the leaf level controls services at a place, while a server at a higher level aggregates information on the lower level servers. Users and services authenticate with servers for service lookups and registration, respectively. SSDS is one of the first secure service discovery protocols and provides many security features. Nevertheless, privacy concerns may hinder users and service providers to expose information to the central servers. For example, Bob's MP3 player refreshes its service registration every five minutes, and thus Bob's daily itinerary is known to the servers. Unlike the trusted central server approach, we assume 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 [13], users and service providers exchange one-time code words to discover each other's existence. By encoding code words in a special form, a user sends one network packet to include code words for all service providers from who he acquires credentials. If a service provider finds a match on a code word, he replies with another one-time code word for the user to verify, otherwise, the service provider keeps silent. The procedure of exchanging code words is shown in Figure 1. More specifically, a code word is generated from a shared secret between a user and a service provider using a hash function. A code word is represented as one bit in an array and it is very high in probability that only legitimate parties can generate and verify the code word. Based on the matched code words, proper credentials are selected and submitted to the respective service providers for authentication. Then, a user queries service providers for services. In short, only legitimate parties gain access to sensitive information.

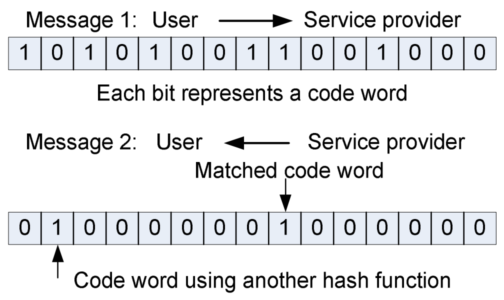
Message 1: User → Service provider
1 0 1 0 1 0 0 1 1 0 0 1 0 0 0
Each bit represents a code word
Message 2: User ← Service provider
Matched code word
0 1 0 0 0 0 0 0 1 0 0 0 0 0 0
Code word using another hash function

Figure 1. A user and a service provider exchange code words in PrudentExposure.

Similar to PrudentExposure, in this paper, a user utilizes a program to manage all his credentials; and users and service providers exchange code words. Thus, the discussion of the credential management program and the properties of code words will not be repeated in this paper. Unlike PrudentExposure, the progressive approach uses a different method to exchange code words and service information. More important, it uses a stronger criterion to protect sensitive information and avoid unnecessary exposure to legitimate parties. Instead of authenticating and querying all service providers that a user has credentials, a user only authenticates with service providers that have the service and excludes the rest of service providers during the process. Likewise, a service provider uses both the user's and service's legitimacy as criteria when exposing his information. Figure 2 illustrates the different design goals of PrudentExposure and the progressive approach from a service provider's point of view. From a user's point of view, the diagram is similar. PrudentExposure discovers legitimate users and service providers. The progressive approach discovers a smaller set: legitimate users and services or legitimate service providers and services.

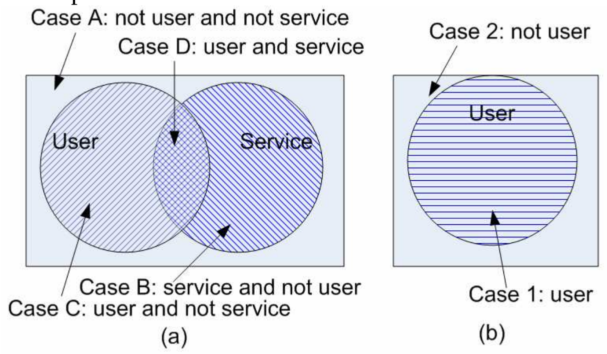
Case A: not user and not service
Case D: user and service
User
Service
Case B: service and not user
Case C: user and not service
(a)
Case 2: not user
User
Case 1: user
(b)

Figure 2. Different design goals of PrudentExposure and the progressive approach from a service provider's point of view. (a) The design goal of the progressive approach is to properly find Case D. (b) The design goal PrudentExposure is to find Case 1.

Other work also influences our approach. Automated trust negotiation systems, such as [14] and [15], 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, while the other party may require seeing a certificate from a Good Business Bureau first. The systems cannot establish trust when there is a conflict, such as the conflict that we are trying to solve.

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 reasonable time. In our approach, a user and a service provider establish trust via a sequence of mutual exposures.

3. A Progressive Exposure Approach

Our approach handles the four cases (in Figure 2 (a)), which have different security and privacy requirements. For Case A and B, a service provider's identity and service information need to be protected because a user is not legitimate. For Case C and D, a user is legitimate. Whether exposing available services and presence information to the user is based on the legitimacy of the service request (the service provider offers the service to the user). Similarly, from a user's perspective, the legitimacy of a service provider and whether a legitimate service provider has the requested service need to be discerned. However, in the beginning, neither a user nor a service provider knows which case it will be. More importantly, we need to solve the chicken-and-egg problem and protect the sensitive information as we discussed in the Introduction.

To solve the chicken-and-egg problem, a user and a service provider expose only partial information (several bits) in turn. 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 both sides, the legitimacy of the user or the service provider and the service will reach a high probability and the communication will stop. When a mismatch occurs, only partial information is exposed and both parties acquire sensitive information with uncertainty. Neither a user nor a service provider can discern Case C from Case A and B when a mismatch happens.

Our progressive approach specifies how users and service providers exposed their information. We illustrate how to protect sensitive information from illegitimate parties. Moreover, since we also protect sensitive information among legitimate parties via uncertainty, an analysis of the exposure in terms of the probability is given and the probabilities are known for each message in the communication. Because false positive matches happen when a discovery message turns out to be Case A, B, or C, we will analyze the expected waste of communication for those cases.

3.1. Expose Sensitive Information Only to Legitimate Parties

During the discovery process, there are two parts of the sensitive information that we protect from illegitimate parties: a user or a service provider's identity and service information. To protect identities, a user and a service provider speak one-time code words. A code word is generated from a time variant parameter (TVP) and a shared secret. Specifically, we use hash-based message authentication codes (HMAC) proposed in [18]:

h(Secret, XOR padding1, h(Secret, XOR padding2, Time Variant Parameter)), where h is MD5.

Unlike PrudentExposure in which a code word is a bit, the code word is a sequence of bits in multiple messages, i.e., only a partial of the hash result is exchanged in each message. The left side of Figure 3 illustrates the generation of a code word. The nature of the HMAC ensures that without knowing the shared secret, it is computationally difficult to find the hash result [19]. Thus, only a legitimate user or a legitimate service provider can correctly generate and verify the code word.

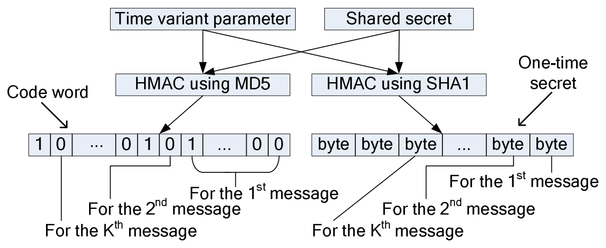

Figure 3. Generating one-time code word to identify existence information (left side); and generating a one-time secret to protect service information (right side).

Services are identified by their hash, and thus service names have the same length. To protect a user's service request and available services of a service provider, a one-time secret is generated at both sides from the shared secret as shown on the right side of the Figure 3. In each message, a user and a service provider use a byte to encrypt or decrypt a few bits of the service information. To be precise, encryption is $cipher = service \oplus one\text{-}time\ secret$ and decryption is $service = cipher \oplus one\text{-}time\ secret$. The encryption method is known as the Vernam cipher [19]. According to [19], if the bytes that we use to encrypt service information is random, our encoding method is computationally secure. We show our tests of the hypothesis in Section 4.1.

3.2. The Protocol

We start with a simplified case that a user interacts with one service provider. The exposure process is shown in Figure 4. First, a user generates one-time code words and secrets. Next, he sends a piece of the code word and service request to a service provider along with a TVP. Then, the service provider generates the code word and secret. If the service provider does not find a match on the code word, he keeps silent. Otherwise, the service provider checks whether there is a match for the service. If he does not find a match, he keeps silent. Unless he finds both matches, the service provider returns 1 or 2 bits of the code word and 1 or 2 bits of available services. Similarly, if there are matches for the code word and the service, the user sends another fraction of the code word and the requested services. 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 authenticate for service access. The numbers of the initial bits and subsequent bits are described in Section 3.3 and 3.4, respectively.

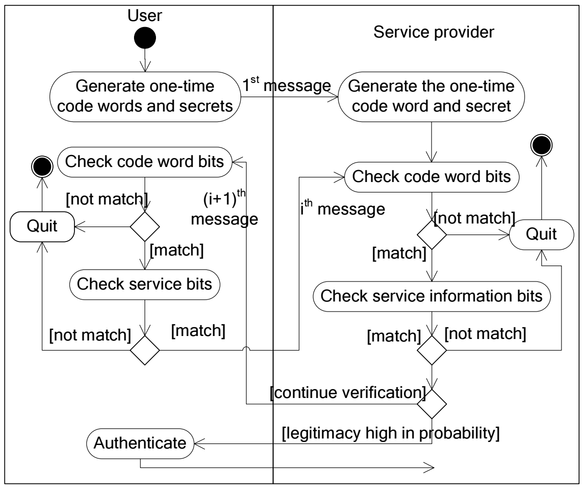

Figure 4. The activity diagram of the discovery process between a user and a service provider.

At a service provider's side, more than one service hashes may match the first several bits of the requested service hash. Therefore, it is possible that the next bit or two bits may be more than one possibility. For example, the service information is 0100 in a user's request; a service provider has two services that start with 0100 and the following bit is 0 for one service and 1 for the other. To inform the user that there are more than one service matches his requested service bit sequence, we encode the possible combinations of the bits as shown in Table 1. If a service provider replies with more than one possible service, a user in his message should indicate which bit or two bits match together with the next bit or two bits.

Table 1. The encoding scheme for service information.

Next single bit	Coding	Next 2 bits	Coding
0	00	00	0000
1	01	01	0001
0 and 1	10	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

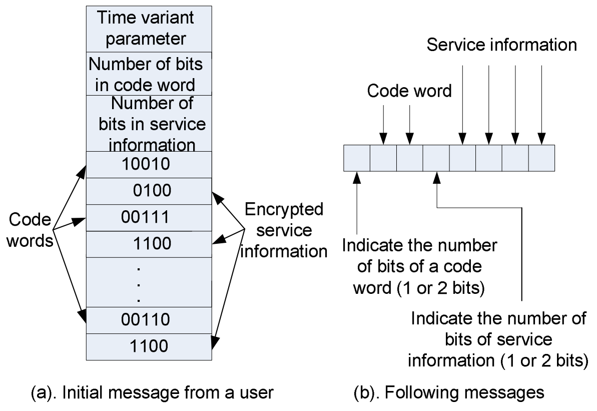
Time variant parameter
Number of bits in code word
Number of bits in service information
10010
0100
00111
1100
00110
1100
Code words
Encrypted service information
Service information
Code word
Indicate the number of bits of a code word (1 or 2 bits)
Indicate the number of bits of service information (1 or 2 bits)
(a). Initial message from a user
(b). Following messages

Figure 5. Message formats.

Without knowledge of the existing services and service providers, a user may specify all code words and encrypted service information in the first message. Figure 5 (a) shows the message format. Since a code word and a service request occupies up to 3 bytes, a user may include hundreds of pairs of code words and service information in one network packet. Then, the message may be sent as a broadcast message or as a multicast message for minimum configuration overhead. The following message between a user and a service provider may be sent via TCP unicast to guarantee message delivery. Figure 5 (b) illustrates the format for the following messages, three bits for code word and five bits for service information. Only the last five bits are encrypted using a one-time secret.

Figure 6 shows the protocol. After first round, a user and a service provider send messages in the same format as message 3 and 4 in Figure 6, respectively. When the discovery process ends, either message A or B is sent to indicate the discovery results.

Since only several bits are exchanged in the beginning, the code words for different service providers may have the same last several bits, called code word conflicts. The probability of a conflict is very high, but the expected numbers of the conflicts are small. It can be proved that:

$$E(codeword\ conflicts) = number\ of code\ words - (1-(1-\frac{1}{2^{numberof\ bits}})^{numberof\ codewords}) \times 2^{number\ of\ bits}.$$

When generating code words, if a user finds the last 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 code words unique.

Notation:
U is a user; S is a service provider.
KUS is a secrect shared between U and S.
KUSI is a secret that U and ith S use to encrypt and decrypt messages.
t_X is a timestamp that X attaches.
R_X is a random number that X generates.
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.
CB^J_{SUI} is bits of a code word shared between U and the ith S in the Jth message.
K_{SUI} is a symmetric encryption key generated at U and the ith S. $\{\}^N$ is a set of N elements.
Q is a message indicates the communication stops.
A is a message instructs a user to authenticate.

No.	Sdr/Rvr	Message
1	$U \rightarrow S$:	R_U, t_U, $\{CB^1_{SUI}, (SRB^1)_{KUSI}\}^N$
2	$S \rightarrow U$:	R_U, t_U, CB^1_{SU}, CB^2_{SU}, $(SAB^2)_{KUS}$
3	$U \rightarrow S$:	R_U, t_U, CB^3_{SU}, $(SRB^3)_{KUS}$
4	$S \rightarrow U$:	R_U, t_U, CB^4_{SU}, $(SAB^4)_{KUS}$
A	$U \rightarrow S$: or $S \rightarrow U$:	R_U, t_U, Q
B	$S \rightarrow U$:	R_U, t_U, A

Figure 6. The protocol.

3.3. Predictable Exposure

We now examine matches of the code words quantitatively during the discovery process. When verifying code word bits, a service provider finds either a match or a mismatch. If a mismatch occurs, he knows the user is illegitimate. If a match happens, he does not know whether the user is legitimate or a false positive match occurs. A service provider is interested in the probability that given a match is found in a message, what is the probability that the message comes from an illegitimate user, namely $p(not\ user \mid match)$. 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 legitimate user, $p(user)$.

The first probability, $p(match \mid not\ user)$, depends on the design of our approach: the number of code words a user has and the number of bits exposed so far. Assuming that the hash results of the last several bits follow the Integer distribution (all possible values are equally likely), the probability in the first message is:

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

Given a message is not from a legitimate user, we want to control the false positive match and still preserve the uncertainty. Thus, we may set the limit to 25%. A user may simply select the number of bits to expose from 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. Afterwards, the false positive rate will decrease by half for each message.

Table 2. Number of bits to expose in a code word vs. the number of credentials a user has.

No. of bits	4	5	6	7	8	9	10
Number of credentials	<5	<10	<19	<37	<74	<148	<295

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%. Before a service provider accumulate enough history information, he may use a fixed strategy, for example, always exchange up to 5 message rounds.

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 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 $p(not\ service \mid match)$. (Note the match here means that a service provider finds matches on the service information.) It depends on three facts: the probability that the service provider has the service, $p(service)$, the probability of false positive matches, $p(not\ service \mid match)$, and the number of services a service provider has. Since a user does not know how many services a service provider has at the discovery moment, the user may by default send four bits of the service information. $p(not\ service \mid match)$ may be calculated at the service provider's side and is very similar to the calculation of $p(not\ user \mid match)$. In addition, a service provider learns the $p(service)$ from history information.

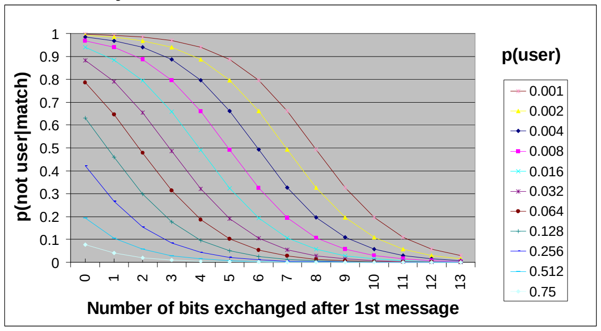

Figure 7. $p(not\ user \mid match)$ **decreases as the number of code word bits exposure increase after the 1st message.**

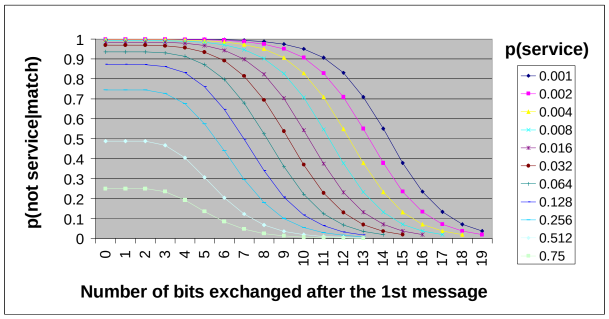

Figure 8. $p(not\ service \mid match)$ **decreases as the number of service information bits exchanged increases after the 1st message. (A service provider with 320 services.)**

We graph the relation between $p(not\ service \mid match)$ and number of bits exchanged after the first four bits for a service provider with 320 services in Figure 8. If a service provider has 160 (or 640) services, he needs to exchange one less (or more) message to reach the same probability.

Therefore, in each round a service provider knows the probability of legitimacy of a user and a service request. Moreover, a service provider may choose critical values for $p(not\ user \mid match)$ and $p(not\ service \mid match)$, for example 5% for both probabilities. During a discovery process, if the probabilities are less than the critical values, the service provider thinks that legitimacy is high enough and therefore finishes the discovery process.

3.4. The Exposure Strategies

During a discovery process, a service provider does not need to calculate the probabilities to determine whether the legitimacy of a user and the user's service request reaches a high probability. Instead he only needs to perform table lookups. Because once the critical values are decided, the numbers of bits that are necessary to reach the critical values are derived directly from the calculation results as we discussed in Section 3.3. Table 3 (a) lists the number of bits for different $p(user)$ values, and Table 3 (b) lists the number of bits for different $p(service)$ values and different number of services for critical values at 5%. 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. When the number of registered services, $p(user)$, or $p(service)$ changes, a service provider performs a table lookup.

Table 3. 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)

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 four combinations: 1/1, 1/2, 2/1, and 2/2 (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 three reasons. First, when mismatches occur, exchanging 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. Third, 2/2 is used to make the convergence of the discovery process quicker when the two numbers of bits are large. The disadvantage of the progressive exposure process is that the number of messages required to reach critical values may be large. For example, 20 bits require at least 10 messages to be exchanged. However, our experiments show one message only takes about 4 milliseconds on a PDA.

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 knows how many bits to exchange in a reply message. 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 is from the service provider's perspective. From the user's perspective, he may trust a service provider's strategy. If a service provider is illegitimate, 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. If a service provider is legitimate 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 that authentication (his identity exposure) is necessary. If a service provider is legitimate but he does not provide the requested service, he knows the user's identity and the service request more accurately by exchanging messages more than necessary. However, the service provider also exposes his identity more precisely. Moreover, to learn more about user's sensitive information, he needs to claim that he has the service by correctly guessing the next 1 or 2 bits of service information or claim that he has multiple services that match the initial sequence of the service hash bits. The behavior pattern can be detected if the service provider does it many times.

3.5. False Positive Match Overhead

The overhead of false positive matches can be calculated from the probabilities that we discussed in Section 3.3. Consider the exposure process as a Markov chain illustrated in Figure 9. For each transient state (state *"0"* to *"k"*), if both the bits of a code word and service information match, the process goes to the next state. Otherwise, the process goes to the absorbing state, *"Quit"*. Given that a discovery message is not legitimate (the user is illegitimate or the user is legitimate but the service provider does not have the service), the process should go to the "quit" state. Since there are false positive matches, the process may go to the next state.

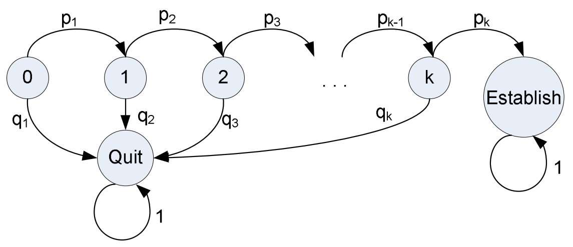

Figure 9. Message exchange process expressed as a Markov chain.

Suppose given a discovery message is illegitimate, 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:

$$p_i = (p(match \cap not\ user) \times p(match \cap not\ service) - p(match \cap not\ service)) \div (p(not\ user) + p(not\ service))$$

Moreover, based on the calculation of the mean time spent in transient states, we calculate the probabilities that the Markov chain makes a transition into state "*i*" given it starts from state *"0"* [20]:

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

$$f_{0,j} = \frac{s_{0,j} - \delta_{0,j}}{s_{j,j}} \quad \text{where } \delta_{0,0} = 1 \text{ and } \delta_{0,j} = 0 \text{ when } j \neq 0,$$
$$\text{and } f_{0,j} \text{ is the probability in state } j \text{ given it starts in "0".}$$

For example, if a service provider has 80 services, $p(user)$ is 0.016, and $p(service)$ is 0.032, and he selects the strategy 1/2, 1/2, 1/1, 1/1, 1/1, 1/1, 1/1, 1/1, 1/1 and 1/1, $f_{0,j}$ is shown in Table 4. Thus, the false positive match overhead decreases quickly.

Table 4. Probability in states given a discovery message is not legitimate.

States	0	1	2	3	4	5	6	7
$f_{0,j}$	0	0.6143	0.2451	0.0347	0.0026	9.7e-5	1.9e-6	1.79e-8

4. System Evaluation

In this section, we first show tests of hypotheses upon which our approach is based. Then, we measure the performance of our protocol.

4.1. Hash Results Follow the Integer Distribution

Throughout the discussion in Section 3, we assume that the last dozens of bits of hash results follow the Integer distribution. For service information, we may encode services to evenly distribute among the first several bits. For code words, however, the good security properties are important and should be retained. On the other hand, the probability of $p(match \mid not\, user)$ is based on the Integer distribution assumption. Moreover, $p(not\, user \mid match)$ and $p(user)$ are based on $p(match \mid not\, user)$. If the assumption does not hold, the abovementioned probabilities are affected. Therefore, exposures may not be predicted well. In addition, we also assume that the last 5 bits of each byte in the one-time secrets follow the Integer distribution. Otherwise, our encryption method for service information may not be computationally secure. Thus, the null hypothesis of the first test is that the last dozens of bits in code words follow the Integer distribution and the null hypothesis of the second test is that the last 5 bits of every byte in a one-time secret follow the Integer distribution.

We use the chi-square goodness-of-fit test to determine if the data can be adequately modeled by the Integer distribution [21]. 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 that was discussed in Section 3.1 to generate a large number of one-time code words and secrets. Then, we count the number of occurrence for each outcome. Last, we calculate the chi-square test statistics and select 5% as the significance level.

For the first hypothesis test, 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 and one of the 20 tests is significant. 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. Similarly, we generate 20 groups of 10,000 secrets to test byte number 6 again and no p-value is significant in the tests. Therefore, we do not reject the null hypothesis.

4.2. Experimental Results

Users and service providers may access or provide services via portable devices, which have limited computing power and energy. Thus, we measure the performance of our protocol on Compaq iPAQs. 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.

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 time. When a user generates 100 code words, a sophisticated version that generates unique code words as we discussed in Section 3.2 is used. The discovery process between a user and a legitimate service provider who provides the service takes about 100 milliseconds. Therefore, within reasonable time, a user can finish the discovery process.

Table 5. Performance measurement of the protocol.

Party	Operation	Time
User	Generating 100 one-time code words and secrets, and sending discovery messages	55.64ms
User	Waiting time from sending the first message to receiving the first reply	6.62ms
Service provider	Generating the code word and secret, verifying all code words and secrets	2.96ms
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

5. Conclusion and Future Work

In this paper, we identified that during service discovery in pervasive computing environments, sensitive information not only needs to be protected from illegitimate parties to access, but also should be exposed only to necessary legitimate parties. We designed a progressive approach to secure sensitive information and determine necessary exposures. We presented the analysis of the mathematical properties, and performance measurement of the protocol.

Currently, a secret is shared between a service provider and his users. We are working on supporting a secret that is shared between a service provider and an individual user. Thus, fine-grained access control rules may integrate with the discovery process. However, supporting a shared secret at the user level causes the processing at a service provider's side to be more complex. Multiple code word matches may be found for different users in the first message and might cause the process to diverge. We are also working on a revocation mechanism. Revocation of shared secrets from a user may not be simple, because a new secret needs to be distributed to other users.

Our approach does not have a specific mechanism for situations when many users and service providers are in a given place, for example, a stadium. A discovery process will cause many false positive matches and waste computation resources and energy. Moreover, our approach may be inefficient when a user knows the existence of a service and its associated service provider, such as repeated discoveries in a short time period. We would like to improve our design to support these situations. A possible solution is to integrate with more precise discovery approaches, such as the approach used in PrudentExposure. However, the rules to determine when to use a precise approach and when to use a progressive approach could be complex.

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.

Acknowledgments

We thank Lyudmila Sakhanenko for formal proof of the expected number of code word conflicts described in Section 3.2, and James Stapleton for discussing the hypothesis testing approaches described in Section 4.1.

References

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

[2] F. Zhu, M. Mutka, and L. Ni, "Classification of Service Discovery in Pervasive Computing Environments," Michigan State University, East Lansing MSU-CSE-02-24,available at http://www.cse.msu.edu/~zhufeng/ServiceDiscoverySurvey.pdf, 2002.

[3] Microsoft Corporation, "Universal Plug and Play Device Architecture," Version 1.0 ed: Microsoft Co., 2000.

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

[5] 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.

[6] 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.

[7] C. Ellison, "Home Network Security," Intel Technology Journal, vol. 06, Issue 04, pp. 37-48, 2002.

[8] Salutation Consortium, "Salutation Architecture Specification," The Salutation Consortium Inc.available at ftp://ftp.salutation.org/salute/sa20e1a21.ps, 1999.

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

[10] Bluetooth SIG, "Specification of the Bluetooth System -- Core," available at http://www.bluetooth.org/docs/Bluetooth_V11_Core_22Feb01.pdf, 2001.

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

[12] 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.

[13] 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.

[14] P. Bonatti and P. Samarati, "Regulating service access and information release on the Web," presented at 7th ACM conference on Computer and communications security, Athens, Greece, 2000.

[15] T. Yu and M. Winslett, "A Unified Scheme for Resource Protection in Automated Trust Negotiation," presented at 2003 IEEE Symposium on Security and Privacy, Oakland, CA, 2003.

[16] M. Krzywinski, "Port Knocking: Network Authentication Across Closed Ports," in SysAdmin Magazine, vol. 12, 2003, pp. 12-17.

[17] T. Pering, M. Sundar, J. Light, and R. Want, "Photographic Authentication through Untrusted Terminals," IEEE Pervasive Computing, January-March, pp. 30-36, 2003.

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

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

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

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