# SmokeScreen: Flexible Privacy Controls for Presence-Sharing

Landon P. Cox, Angela Dalton, and Varun Marupadi
Department of Computer Science, Duke University
Durham, NC, USA
lpcox@cs.duke.edu, angela@cs.duke.edu, varun@cs.duke.edu

## ABSTRACT

Presence-sharing is an emerging platform for mobile applications, but presence-privacy remains a challenge. Privacy controls must be flexible enough to allow sharing between both trusted social relations and untrusted strangers. In this paper, we present a system called SmokeScreen that provides flexible and power-efficient mechanisms for privacy management.

Broadcasting *clique signals*, which can only be interpreted by other trusted users, enables sharing between social relations; broadcasting *opaque identifiers (OIDs)*, which can only be resolved to an identity by a trusted broker, enables sharing between strangers. Computing these messages is power-efficient since they can be precomputed with acceptable storage costs.

In evaluating these mechanisms we first analyzed traces from an actual presence-sharing application. Four months of traces provide evidence of anonymous snooping, even among trusted users. We have also implemented our mechanisms on two devices and found the power demands of clique signals and OIDs to be reasonable. A mobile phone running our software can operate for several days on a single charge.

## Categories and Subject Descriptors

D.4.4 [**Operating Systems**]: Communications ManagementNetwork communication; D.4.6 [**Operating Systems**]: Security and Protection—*Access Control*; D.4.7 [**Operating Systems**]: Organization and DesignDistributed systems; D.4.8 [**Operating Systems**]: PerformanceMeasurements; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

## Keywords

Location privacy, mobile computing, pervasive computing, presence-sharing, social networks

## General Terms

Human Factors, Measurement, Security

## 1. INTRODUCTION

Presence-sharing is a cooperative service provided by independent, mobile participants periodically broadcasting their identity via short-range wireless technology such as BlueTooth or WiFi. In combination with location services like GPS and PlaceLab [23], these networks provide a promising platform for emerging mobile and pervasive applications. Examples include automatic tagging of mobile data (e.g. digital images or lecture notes) to support file search [7], mobile social networks [10], and messaging services like the "missed connections" feature of CraigsList [7].

Presence-sharing is an attractive alternative to more traditional pervasive computing architectures because of its low cost, decentralized approach, and ease of deployment. For example, projects such as the ContextCam [34] can be used to automatically tag digital images, but require a large investment in sensor and RFID technologies.

Despite the advantages of presence-sharing, privacy protection remains an important unmet challenge. Mobile users are unlikely to participate if anonymous strangers can roam through a crowd collecting identities or marketers can surreptitiously compile users' location histories. Operating in a decentralized wireless environment makes managing presence-privacy particularly difficult. Wireless broadcasts can be heard by anyone, including unwanted snoopers. Thus, broadcasts must be constructed so as to only reveal identifying information to a subset of users.

A simple protocol that meets this requirement is one in which members of a trusted group initially negotiate a secret symmetric key. To later reveal themselves, members broadcast their identity and a nonce encrypted with the secret key. Message recipients use their secret key to verify that the decrypted content matches a member of the group. The drawback of this approach is that it is *inflexible*: users cannot tune their access control policy to their social environment nor can they safely share their presence with strangers.

For example, in a file-tagging service users may want to restrict sharing to social relations such as friends, family members, and co-workers. However, trust in location privacy is a function of fluid social dynamics and users must be able to adjust their access control policies accordingly. Studies of attitudes toward location privacy have revealed adults who trust their co-workers with their location during business hours but not otherwise [6], teenage girls who trust their parents except when they are socializing with friends [27], and spouses who trust each other except when shopping [6].

In addition, restricting presence to established social relations cripples many useful presence-sharing applications. Within a mobile social network users will likely want to see the presence of both friends and like-minded strangers. For missed connection messaging, users must be able to send messages to people they have encountered but do not know.

In this paper, we present a privacy management system called *SmokeScreen* that addresses these issues though two complementary mechanisms. To enable flexible sharing between social relations, users broadcast *clique signals* which can be activated or deactivated depending on the social environment. To enable flexible sharing between strangers, users also broadcast *opaque identifiers* (OIDs) which are time-, place-, and broadcaster-specific and can only be *resolved* to an identity via a centralized trusted broker. Together these mechanisms allow users to both manage their location-privacy and reap the full benefit of presence-sharing. Furthermore, these mechanisms are power-efficient since devices can avoid broadcast-time cryptographic work by pre-computing and comfortably storing 48 hours worth of future signals and OIDs.

The rest of this paper is organized as follows: Section 2 motivates the need for flexible privacy controls by describing some representative presence-sharing applications and summarizing a prior study of user attitudes toward location privacy; Section 3 describes SmokeScreen's trust and threat model; Section 4 describes SmokeScreen's design; Section 5 describes the SmokeScreen prototype; Section 6 evaluates the prototype and examines users' behavior in a deployed presence-sharing application; Section 7 and Section 8 discuss related work and provides our conclusions, respectively.

## 2. BACKGROUND AND CONTEXT

Presence-sharing is a cooperative service among mobile devices that provides applications with the identities of co-located users. More concretely, participants use discovery protocols such as those provided by BlueTooth or WiFi to scan for other co-located devices. The set of discovered MAC addresses and device names can then be mapped onto a set of identities by higher layers of software. For example, a simple presence-sharing scheme might consist of users including an email address in their mobile phone's BlueTooth device name.

### 2.1 Presence-sharing Applications

These networks can be used to support a number of useful applications. One such application is automatic file-tagging. Both commercial [13, 37] and research [39] operating systems have begun to embrace search as a first-class data organization tool. These systems typically rely on file names, timestamps, embedded keywords, and computational context [39] to build their search indexes. Unfortunately, there are occasions when conventional sources of meta-data are inadequate or unavailable.

One such occasion is when a user wants to retrieve media files based on the identity of their subject, such as "find the pictures of Bob." Support for this kind of search is both important and hard. The portion of PC users' data composed of personal multi-media files such as digital photographs and video is growing exponentially [28] and at least one study of camera phone use found that images of people comprised over half of all images taken [21]. Furthermore, media files are notoriously difficult to index since they do not contain text and are assigned opaque names by the devices that create them.

Presence-sharing can automatically assign searchable, identity-based attributes to images by associating the presence information available when a picture is taken. MMM2 [8] comes closest to this idea by using presence to suggest users with whom to share an image.

Presence-sharing can also be used to support mobile social networks. Traditional social network websites, such as Facebook, Friendster, and MySpace, allow users to create personalized profiles that are linked to their friends'. This allows users to find people with common interests by browsing the profile graph. A mobile social network network provides similar opportunities to meet new people only in physical space rather than over the Internet.

Social Serendipity [10] typifies such a network. Each Serendipity user fills out a profile containing a small photograph, interests, username, and list of friends. Serendipity also associates each profile with a BlueTooth MAC address and mobile phone number. In social situations, a Serendipity server called BlueDar listens for nearby BlueTooth devices and uses the devices' MAC addresses to look up their associated profiles. If there are co-located users with overlapping interests, Serendipity sends a text message containing the profile of each user's match along with a suggestion that they meet. Based on the profile's photo, a user may then look for their match in the room and introduce themselves.

Finally, our prior work described how presence information can be used to support "missed connection" messaging among mobile users [7]. The term "missed connection" is derived from a feature of the popular website CraigsList. This service allows users to post messages for people they encountered in the recent past but were unable to speak to at the time. Cities such as Boston, New York and San Francisco generate hundreds of missed connections on craigslist.com each day. Most postings are romantic inquiries, but there are also requests for lost items, such as "did anyone find the laptop I left in my taxi around 2PM," and notifications of found items, such as "I found a set of keys at the coffeeshop."

While popular, this service is by no means ideal. First, users can never be sure if their message has been seen by the intended recipient. Second, authenticating respondents can be difficult. It is not uncommon to see messages in which a respondent is required to provide some detail of the encounter, such as "You were my waitress, please tell me what I ordered." Presence-sharing along with a trusted service mapping devices to profiles could improve both problems.

Users could record the identities of the devices they came into contact with. Then based on the presence information recorded during a social setting, they could ask the service to route messages to the owners of each device. To further ensure that the message reached the correct target, the sender might be allowed to browse the profiles corresponding to the devices she saw or specify that the message only be forwarded to users with specific attributes (male or female, older or younger, etc.). This design could also support messaging users with indirect links. For example, a taxi driver's presence could be used to connect the owner of a lost laptop to the passengers that followed him.

### 2.2 Location Privacy

In addition to the potential benefits of presence-sharing there are also many plausible scenarios in which presence-information could be abused. These can range in severity from the inconvenience of unwanted advertising to the danger of high-tech stalking. Though it is impossible to know how likely these abuses will be, system designers can still ask users what their concerns are.

There have been many studies of users' attitudes toward technology and privacy [24, 27, 33], but researchers from Intel's Seattle Research Lablet recently carried out a particularly revealing one [6]. In this study, participants provided a list of up to 17 social relations and categorized each based on the nature of their relationship, such as a family, co-worker, or friend. Participants were then given a mobile device for two weeks, to which the researchers sent 10 randomly timed location requests from their social relations each day.

There are three noteworthy features of the study's methodology. First, the study observed users in normal social settings. A weakness of some previous user studies has been their focus on work environments [3]. Second, users experienced an actual system for two weeks. Thus, the observed behavior is more likely to reflect users' actual attitudes than survey results alone. Third, while the study's system differs from a presence-sharing network in some ways (e.g. presence-sharing assumes co-located users, but could conceivably support remote queries), it is similar enough to provide insight into the concerns that presence-sharers might have.

One of the key results is that 23% of location requests were denied, even though requests came exclusively from users' list of social relations. In the words of the authors, these denials were generally used to "reinforce or communicate social boundaries." For example, one participant felt that it was inappropriate for coworkers to know his location outside of business hours. Other denials include friends' requests during work hours and spousal requests while doing something the user should not have been doing, such as shopping or resting rather than walking the family dog.

Users also had more unpredictable reasons for rejecting requests. One user would only reply to requests from her mother when she was not drinking alcohol. Some refused to disclose their location because they were angry with the requester. For example, one user refused a request from his mother because they had recently fought. These results support the view that presence-sharers need the flexibility to tune to their access controls to their social environment.

Importantly, users' privacy concerns were not limited to social relations. Over half of the participants (9 out of 16) expressed more general concerns about the effect of location services on their privacy. One user noted that thieves could benefit from knowing where he was. Others used phrases like "being on a leash," "stalking," "being monitored," and "Big Brother" to describe their unease. These responses affirm our belief that anonymous snooping is also a common concern for users. The central challenge for SmokeScreen is to provide support for a diverse set of presence-sharing applications without compromising users' privacy.

## 3. TRUST AND THREAT MODEL

Trust in SmokeScreen is split into two categories: 1) users may be trusted to handle shared cryptographic state such as secret keys, and 2) users may be trusted to record a user's presence at a given place and time. The first category of trust is symmetric and established among a group of social relations. The second category is potentially asymmetric and depends on the social context of a particular time and place.

Importantly, neither trust category implies the other. A user may establish a shared, secret key with a social relation, but never wish for that social relation to know where they are. Alternatively, a user may wish for a stranger without a pre-established trust relationship to be aware of their presence, as in a mobile social network.

In addition to these trust relationships, all SmokeScreen users trust a key distribution infrastructure and brokering service. Each user has a single unique identity, defined by their public key; this prevents SmokeScreen from being vulnerable to Sybil attacks. Social networking websites such as Facebook [11] and MySpace [26] can be used to distribute keys, and Section 4.2.2 describes our trusted brokering architecture.

Given these trust parameters, SmokeScreen defines an adversary as anyone who is not trusted to record a user's presence at a particular time and place, even though the adversary may be present themselves. Of course, SmokeScreen cannot prevent presence information from leaking though "hidden-channels" such as face-to-face contact or a car's license plate.

SmokeScreen is resilient to collusion among adversaries, but is vulnerable to collusion between adversaries and trusted users. We assume that adversaries are capable of compiling complete broadcast histories at a particular location and can share broadcast histories with one another. However, SmokeScreen cannot handle unauthorized sharing of secret keys or presence information by trusted users with adversaries.

SmokeScreen also cannot prevent an adversary from detecting a user's absence. If a user leaves a location, SmokeScreen will not continue to broadcast their presence information. It is unclear how useful detecting a user's absence would be. If an adversary is unable to detect a user's presence it could be because the user is physically not present. However, it might also be that the user is physically present but in non-discoverable mode or present but has decided to stop broadcasting to the adversary.

Finally, SmokeScreen is currently focused on device information such as BlueTooth device names and WiFi SSIDs that are easy to change in software, but full presence-privacy also requires eliminating the static MAC addresses broadcast by layer two wireless protocols. Though SmokeScreen does not currently provide privacy protection down to the MAC layer, many WiFi cards support MAC address scrubbing [25] and nearly all BlueTooth radios' MAC addresses can be reset via vendor-specific Host Controller Interface (HCI) commands. Prior work on disposable MAC addresses [18] used these techniques to provide location privacy for WiFi users.

Unfortunately, changing MAC addresses can have other consequences. For example, BlueTooth devices often use MAC addresses to authenticate one another and some WiFi cards are not fully functional while operating under a "false" address. Because of this, for many users full location privacy may only be practical during idle device periods. We believe that most devices will be idle in the common case, but a full discussion of the implications of changing MAC addresses is beyond the scope of this paper; we hope to investigate these issues in the future.

## 4. DESIGN

Presence-sharing is attractive for its openness as well as its ease of deployment and the design of SmokeScreen assumes that users are independent, self-interested, and come and go as they please. To address users' privacy concerns without sacrificing these features, we identified four design goals to guide us:

- **Control** Users should retain full control of when and where their presence information is released. Location information should never be revealed without the user's explicit permission.
- **Disclosure** To eliminate anonymous snooping, presence may only be revealed under two conditions: recipients must be known social relations or strangers who become known. Ideally, users could identify potential recipients before revealing their presence, but this may not always be possible. Users should at least be told who viewed their presence after the fact.
- **Isolation** Revealing presence at one moment should not affect past or future access control decisions. For example, clique members must not be able to identify users unless their clique is active. Similarly, a single OID resolution should not provide information about other unresolved OIDs from the same broadcaster.
- **Dispersion** There should be no centralized collection point. Even though the OID broker is a centralized point of coordination between strangers, it must not be able to compile long-term location histories of users. Otherwise, it could become an attractive target for hackers and nosy administrators.

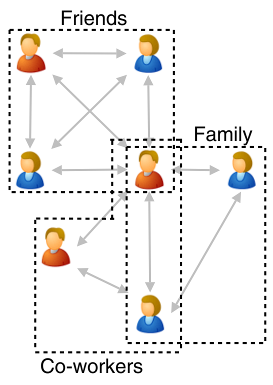


**Figure 1: Minimum clique cover**

With these goals in mind, we separated the problem of presence-privacy into two cases and applied a separate solution to each case: *clique signals* regulate sharing between social relations and *brokered exchanges* regulate sharing between strangers.

Users manage access to their presence among social relations by organizing them into mutually trusted sets of users defined by the minimum clique cover of their social relations. In practice these cliques correspond to natural social categories such as a user's family, college friends, and co-workers [6]. Each clique has an associated secret key which is used to compute a unique *clique signal*.

As users move through the world, they broadcast a set of signals, one for each *active* clique, that notify other users of their presence. Only a clique member can interpret the clique's signal. A clique is active when a user's access control policy allows each clique member to view its presence. Activating and deactivating signals fits access control policies to the social context.

Presence-sharing between strangers requires a different mechanism. Thus, in addition to clique signals, users broadcast opaque identifiers (OIDs). Although time, place, and broadcaster specific, an OID provides no useful information to its recipient unless it can be *resolved* to its broadcaster's identity. To resolve an OID, its recipient must provide a resolution of its own. This condition eliminates anonymous snooping and is enforced by a trusted *broker*.

## 4.1 Sharing Between Social Relations

Users share their presence with social relations at the granularity of a *clique*. Within a clique, each member trusts every other member. The smallest possible clique is a pair of users, though larger cliques arise naturally from users' social networks; all members of a family trust one another as do all members of a group of college friends. We first examine how to discover a user's cliques by computing the minimum clique cover of its social relations and then show how these cliques can be used to regulate presence-sharing.

### *4.1.1 Computing Cliques*

For a user, $u$, with social relations, $R = \{r_1, r_2, \ldots, r_n\}$, a clique cover is a set of subsets, $C_1, C_2, \ldots, C_m$ of $R \cup \{u\}$ such that $u \in C_i$; $C_1 \cup C_2 \cup \ldots \cup C_m = R \cup \{u\}$; and every member of $C_i$ trusts every other member of $C_i$. We assume that trust between social relations is always symmetric.

One very simple cover is the maximum clique cover in which $n = m$ and $C_i = \{r_i, u\}$. Computing the maximum cover is trivial and provides the greatest flexibility for designing access control policies. Unfortunately, it greatly increases the communication costs of signaling other users. For example, Westerners typically have between 10 and 30 social relations [22], which, even for signals such as a 20-byte cryptographic hash [32], on average leads to a 300-byte broadcast. Typical users of online social networking sites such as Facebook often have social networks of over one hundred users [16]. In practice using a signal for each user would be too large for a BlueTooth device name, which is limited to 256 bytes.

Thus, rather than using the maximum clique cover, we compute a minimum clique cover: the smallest $m$ such that $C_1 \cup C_2 \cup \ldots \cup C_m = R \cup \{u\}$. Figure 1 shows an example minimum clique cover. This minimizes the communication overhead of each broadcast. For example, if each $r_i$ were a social relation of every other $r_j$, then the minimum cover would be $C_1 = R \cup \{u\}$ and a single signal could be broadcast.

Another possible complication is churn. If the composition of cliques changes too quickly, then signals may become inconsistent. One option is to recompute the clique cover at a well-known time (say four in the morning) and then assume its consistency until the following synchronization point. The drawback of this scheme is that there will be delays until updates to the graph are reflected among mobile users. However, if clique churn is slow as, is the case in most online social networks, then inconsistency should be rare.

Finally, how users represent their social relations to a software package that can compute the minimum clique cover is beyond the scope of this paper. However, many users will be able to take advantage of existing on-line social networking sites such as Friendster and MySpace. As of February of 2006, MySpace was the 13th most visited site on the web and generated two and half times the traffic of Google [4]. MySpace alone had 54 million users and Friendster had 24 million [4]. We are currently looking into using these graphs to automatically generate clique signals.

### *4.1.2 Computing Clique Signals*

Once a user has computed its minimum clique cover, it needs to compute each cliques' unique *signal*. For example, say that the sequence of members of clique $C$ is $c_1, c_2, \ldots c_n$. Initially, the members negotiate a shared secret key, $K_c$. The clique signal is computed by applying a cryptographic hash [36, 32], $H$, to the secret key: $S_c = H(K_c)$. The cryptographic hash keeps signals small, deterministic, and secure.

Broadcasting signal $S_c$ notifies fellow clique members, but does not uniquely identify the broadcaster. Thus, for each active clique, users broadcast pairs of values, $\langle S_c, \{i, t\}_{K_c} \rangle$, where the broadcaster is the $i$th member of the clique and $t$ is a timestamp to prevent replay attacks. An appealing feature of this construction is that it does not require expensive asymmetric cryptography. Furthermore, these pairs ensure the goals of *disclosure* and *control*. Only clique members can identify the broadcaster of its signal, and activating and deactivating signals allows users to exclude clique members when appropriate.

Unfortunately, we may still be in violation of the goal of *isolation*. Consider a scenario in which user $a$ broadcasts signals, $\langle S_i, m \rangle, \langle S_j, n \rangle$ and that user $b$ is in clique $I$ but not $J$. $B$ still knows that $A$ is a member of clique $J$ even though it may not know the other members. Now say that $A$ deactivates clique $I$ and only broadcasts signal $\langle S_j, n' \rangle$, where $n \neq n'$ because of a new timestamp. If $b$ has other knowledge about clique $J$, such as that it is only composed of two members, it can probabilistically infer $a$'s presence.

To remove this possibility, users must deterministically and independently update their clique secret keys. For example, clique $C$'s secret key at time $t$ might be defined as $K_c^t := H'(K_c^{t-1})$, where $H'$ is a different cryptographic hash from the function used to com-

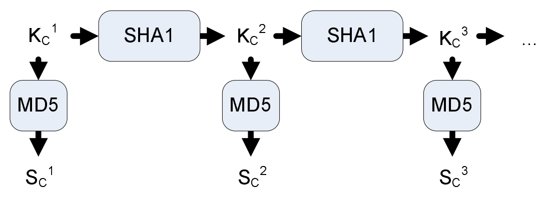


**Figure 2: Clique key and signal generation**

pute the signal, $H$. For example, we currently use `md5` to compute signals and `sha1` to update the secret key. This ensures that signals deterministically change with $t$, but prevents non-members from computing future signals using previous ones. Figure 2 shows this use of hashes.

Users must be synchronized to independently update $t$, but if $t$ only increments each hour or even every minute, users can be loosely synchronized without missing many signals. If synchronization is a problem, users could pre-compute a sequence of signals and compare incoming messages to the pre-computed values.

In fact, a user could conserve the power consumed by broadcast-time cryptography by pre-computing signals. If we assume that signals are 16 bytes (the size of an `md5` hash) and updated each hour, and that the encrypted index and timestamp are also 16 bytes (using a 128-bit AES [5] symmetric key) and updated every minute, then pre-computing 48 hours worth of signals would require just under 45KB of storage per clique. To pre-compute 20 cliques, the total storage requirement would only be 900KB. In addition to conserving energy, this allows users to trade storage space for more frequently updated signals. With mobile phones now commonly providing tens of MB of flash storage, many can use their excess storage capacity to gain better privacy and avoid consuming battery power.

### 4.1.3 Interpreting Clique Signals

The main data structure used to interpret signals is a dictionary that maps signals to the secret key used to compute the signal and a list of clique members. Note that the list of members must be ordered deterministically for indexing to identify the correct user. To improve the space efficiency of the dictionary, the list of clique members only needs to be stored once and can be looked up indirectly through the main dictionary.

When a user receives a signal-index pair, $\langle S_c, \{i,t\}_{K_c}\rangle$, it first looks up the key and list associated with $S_c$. If the lookup fails, the user moves on to the next pair. If the lookup returns key $K_c$ and list $c_1, c_2, \ldots c_n$, the user first decrypts $\{i,t\}_{K_c}$ using $K_c$, verifies that the timestamp, $t$, is fresh, and assuming it is, concludes that user $c_i$ is present.

### 4.1.4 Alternative Constructions

SmokeScreen's clique signal construction allows users to safely share their presence with social relations, but before arriving at it, we considered several alternative constructions. One particularly attractive approach was to try applying a private matching protocol [12] as in the RE email system [14].

Very briefly, one could imagine such a scheme working as follows. Each user could have distributed a different secret key, $K_i$, to each of its $n$ social relations. At time $t$, both the user and her relations could compute a new key $K_i^t$ using cryptographic hash functions as before. The user would then use each of these keys to define a cryptographically encoded polynomial, $P_t$, such that $P_t(K_i^t) = 0$, and broadcast it. If another user received $P_t$, it would then check each of the $K_j^t$ it computed for its social relations. If $P_t(K_j^t) = 0$, then its $j$th social relation broadcast the message. Importantly, the private matching protocol's construction of $P_t$ protects information about its roots, hiding the broadcaster's secret keys from untrusted users. Only social relations could find the roots of $P_t$.

The primary reason why we did not take this approach is that the cryptographic overhead seemed too great for a mobile environment. First, the private matching protocols we found required asymmetric cryptography between the sender and receiver. Not only would this have been expensive, but because the identity of the broadcaster cannot be revealed in plain text, message receivers would have had to try decrypting any incoming messages with the public key of every social relation.

Second, even if the asymmetric cryptography issue could be resolved, private matching protocols would require SmokeScreen users to perform on-line cryptographic work on the order of the size of their social network. For each message received, users would have had to determine whether or not a social relation of theirs was encoded in the broadcaster's message. Adversaries could have exploited this fact to mount denial of service attacks by forcing users to consume extra battery power (again on the order of the size of the victim's social network) through spurious messages. In contrast, SmokeScreen's clique signals require no cryptographic work if the initial lookup fails.

## 4.2 Sharing Between Strangers

If users only want to share with known users, then they only need to broadcast clique signals. However, many presence-sharing applications would be crippled without interactions between strangers. To enable safe sharing within these applications, users can broadcast *opaque identifiers (OIDs)* in addition to clique signals. OIDs reveal no information to their recipient and must be *resolved* through exchanges coordinated by a trusted broker. Our brokered exchange architecture has three components: a mobile device with short-range wireless broadcast capability, an Internet-accessible client process acting on the user's behalf, and the trusted broker. Figure 3 depicts each component.

Users interact with the broker through the client program. The client will most likely run on a PC or workstation that intermittently communicates with the mobile device. However, there is no reason why the client could not run on the mobile device itself to improve resolution latency and provide continuous communication with the broker.

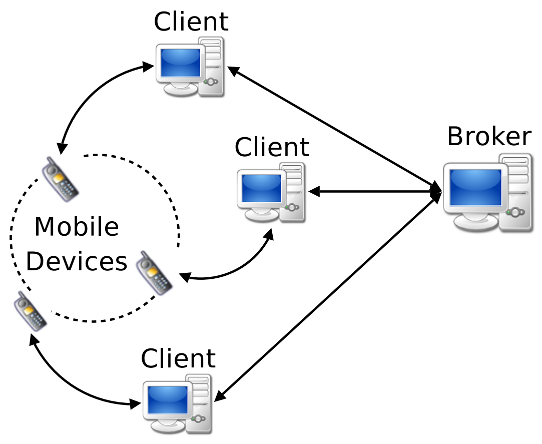


**Figure 3: Brokered exchange architecture**

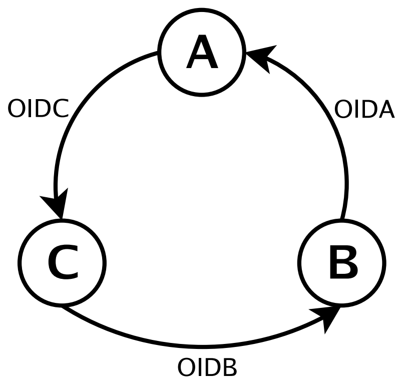


**Figure 4: Example cycle**

### 4.2.1 Opaque Identifiers

The goal of *control* requires that OIDs only be useful once resolved by the broker. Similarly, *isolation* requires that the resolution of a single OID not allow a user to resolve any past or future OIDs. Thus, we define the OID broadcast by user $a$ at time $t_i$ and place $p_j$ to be

$$OID \quad = \quad \{a, n, sig(a,n), \{t_i, p_j\}_{K_a}\}_{pub(Br)}$$

where $n$ is a nonce, $sig(a, n)$ is a digital signature computed over "$a, n$" with $a$'s private key ($pr(a)$), $K_a$ is a secret symmetric key known only to $a$, and $pub(Br)$ is the broker's public key.

Because the OID plaintext is encrypted using the broker's public key, only the broker can know the identity of the broadcaster. Furthermore, because $n$, $t_i$, and $p_j$ change over time, resolving one OID will not help a user resolve any others. This ensures *isolation*.

$a, n$, and $sig(a, n)$ ensure that $a$ created the OID to prevent false identity attacks. The nonce, $n$, prevents signatures from being reused. For example, if $a$ signs a message to $b$ with $sig(a, m)$, then $b$ must not be able to use this signature to generate a false OID. It is possible to replay another user's OID, but the attacker would gain no information as a result since the broker would not include the replaying node in any exchange.

The portion of OID plaintext that is encrypted with $K_a$ is called the *hint*. Hints can contain arbitrary information and are returned to users by the broker as part of the terms of an exchange. Their only purpose is to provide users with information about the context in which an OID was broadcast. Hints also enforce *dispersion*; all location information within the OID is concealed from the broker since only $a$ can decrypt $\{t_i, p_j\}_{K_a}$. The broker only knows who originally broadcast an OID and who is interested in resolving it.

The power consumed by asymmetric cryptography can be conserved by pre-computing OIDs using timestamp-only hints. In such a scheme all OIDs are pre-computed, but whenever an OID is broadcast, the device records $p_j$ in a position log. If a timestamp-only hint were later returned by the broker, the client would use the time to find $p_j$ in the log. Assuming a 128-byte OID and broadcast rate of once per minute, pre-computing 48 hours of OIDs would consume only 360KB of storage, not including the position log. Though we have not implemented this optimization yet, we intend to in future versions of our prototype.

We currently use 1024-bit RSA [31] keys for all asymmetric cryptography. For a BlueTooth device with 256-byte device name, the name must be partitioned into a 128-byte OID (assuming an OID is broadcast) and four 32-byte signal-index pairs. Because more than four cliques may be active at any moment, users must make scheduling decisions about when to broadcast their active clique signals and OIDs. For example, a user might always broadcast an OID and schedule active signals round-robin.

OID_interest

| recipient | OID | bcaster | state | cycleID | vote |
|---|---|---|---|---|---|

pending_trades

| username | terms | retries |
|---|---|---|

pending_votes

| cycleID | result |
|---|---|

user_info

| username | hostname | pubkey |
|---|---|---|

**Table 1: Tables maintained by the broker**

### 4.2.2 The Broker

The broker is fully trusted by all users to coordinate exchanges. It does this in three phases. First, it maintains a graph based on OID interest, in which vertices represent users. Edges are directed from a user who is interested in an OID to the OID's broadcaster and are labeled with the OID. The broker identifies potential n-way exchanges by searching for *cycles* in the graph. Figure 4 shows an example. If the broker finds a cycle, it initiates a two phase commit [15] by bringing the exchange to a vote. Once all the votes have been collected, the broker commits the trade and distributes the result.

Although our broker implementation attempts to identify pairwise exchanges first, we have generalized our approach to support n-way exchanges to preserve flexibility. For example, n-way exchanges do not require exchanges of co-located OIDs. One user may be interested in resolving an OID it received during a summer vacation in California, another may be interested in resolving an OID from a winter conference in the UK, and a third may be interested in resolving an OID from her brother's graduation party in New York. As long as the broker can identify demand between *users*, the OIDs involved could have been collected at any place and any time.

It is also important to note that demand for OID resolutions is entirely application driven. Only if a particular time and place is of interest to an application will the OIDs logged during that period be registered with the broker.

### 4.2.3 Coordinating an Exchange

Because nodes may fail or be unreachable during coordination, the broker maintains its state in four persistent tables: OID_interest, pending_trades, pending_votes, and user_info. Table 1 shows each table's fields. user_info is the least interesting of the four; it only stores the location of each user's client process and its public key.

OID_interest stores the edges of the interest graph. As users register interest in OIDs, the broker inserts new entries into OID_interest using the recipient, OID, and bcaster fields. Edges exist in one of three states: free, locked, or committed. When an edge is first inserted, its state is free, making it available for cycle searches.

After a user registers its interest in an OID (or set of OIDs), the broker reads the edges marked free in the transitive closure rooted at the user into memory. It then performs a breadth-first cycle search on the in-memory copy. If the broker finds a cycle, it assigns a unique *cycleID* by computing the cryptographic hash, such as sha1, over the concatenation of sorted OIDs.

presence

| time | place | OID | bcaster |
|---|---|---|---|

trades

| cycleID | terms | state |
|---|---|---|

**Table 2: Tables maintained by the client**

Once the cycleID has been computed, the broker locks each edge by marking its state in OID_interest as locked and setting the cycleID. If any attempt to lock an edge in the cycle fails, each edge locked up to that point must be unlocked by resetting its state to free and the search abandoned.

With the edges locked, the broker initiates the two phase commit by proposing the terms of the trade to each member of the cycle. Consider the cycle in Figure 4. For user A, the terms of the trade contain the cycleID (e.g. sha1({OIDA, OIDB, OIDC})), the hint embedded in OIDA, the OID that A is interested in resolving (OIDC), and, if possible, for whom A is resolving OIDA (user B). It may not always be possible to provide this final piece of information, as when exchanges only involve two users. We will return to this issue in Section 4.2.6.

Before users can vote on an exchange, the broker must ensure that its vote state remains consistent, even in the face of failure. It does this through the pending_trades and pending_votes tables. Once the broker has computed the terms for each user, it inserts them into pending_trades. When a client has acknowledged receiving the terms, its corresponding entry in pending_trades can be removed. If the broker tries to send a user its trade terms more than some pre-defined number of times, the user's vote is set to "no."

Voting enforces the goal of *control*. If, after considering the terms of an exchange, a user decides against revealing its presence, the broker will try to find a new cycle.

pending_votes contains the current state of all exchanges, where the result field of any entry is initially set to pending. It remains pending until one of two events occur: if all members of the exchange vote in favor, then the vote's result changes to yes; if any member votes against the exchange, then the vote's result changes to no.

If a vote's result is yes, then for each user that has been notified of the result, the broker sets its edge's state to committed. Similarly, if the result is no, then for each user that has been notified of the result, the broker resets its edge's state to free. Once all voters have acknowledged the result, the cycle's entry in pending_votes can be removed.

### 4.2.4 *The Client*

Clients maintain information about the OIDs they have resolved in a presence table and state related to any pending votes in a trades table. Table 2 describes both.

When the client receives an OID or set of OIDs from the mobile device it adds them to the presence table, along with the time and place where the device logged them. Because OID broadcasters are unknown, the bcaster field is initially empty. After inserting new OIDs, the client registers its interest in them and waits for the broker to return any trade terms.

Before acknowledging having received an exchange's terms, the client stores them in the trades table with their state set to free. After reaching a decision about the trade, the client updates its state to yes or no, sends the broker its vote, and waits for the broker to commit or abort the exchange. Before acknowledging an exchange decision, the client updates the identity of the OID broadcaster in presence (if the exchange was committed) and deletes the terms from trades.

### 4.2.5 *Example Two-Way Exchange*

Figure 5 shows the messages involved in a two-way exchange between Alice and Bob. For clarity, we have removed all authentication between users and the broker and combined the functionality of the client and mobile device into a single entity.

Initially, Alice broadcasts $OID1$, which is received by Bob (message 1). Note that Bob remembers when and where he was when he received this message, denoted $t'_i$ and $p'_j$. In message 2, Bob registers his interest in resolving $OID1$ with the broker.

Once decrypted, the broker knows that Alice broadcast $OID1$ and creates an edge from Bob to Alice labeled $OID1$. Note that if Bob registers an interest in $OID1$ shortly after it has been broadcast, the broker could infer with reasonable accuracy when Alice broadcast $OID1$, but not where. Next, Alice receives $OID2$ from Bob and registers her interest in it with the broker (messages 3 and 4).

After the broker has identified a potential exchange between Alice and Bob, it notifies them both (messages 5 and 6). Notification messages include Alice's and Bob's hints, which allow users to reason about their resolutions' sensitivity. Assuming that Alice and Bob agree to the exchange (messages 7 and 8), the broker replies with the users' identities (messages 9 and 10).

### 4.2.6 *Client Cost-Benefit Analysis*

How users will evaluate the trades proposed to them is an important unknown. Unless the broker can identify appealing trades, there will be no sharing between strangers. To avoid this, we want to provide users with as much context about the trade as is safe.

Unfortunately, users cannot be told who will receive their location when there are only two traders. This is fundamental to pairwise trades because identifying a trading partner is equivalent to executing the trade. Of course, once an exchange has occurred, users can be told to whom their information has gone. This satisfies the goal of *disclosure*.

Depending on the application and the exchange, the broker may still be able to give users extra information beforehand. For example, if $a$ is interested in $b$, $b$ is interested in $c$, and $c$ is interested in $a$, $a$ could be given some information about $c$ without disclosing the identity of $b$. In an image meta-data service, users might want a thumbnail of the image they are to be associated with. For mobile social networks, an avatar or picture of the interested user may be an acceptable blurring of actual identity.

In some situations, support for n-way trades may also make it safe to reveal more information about partners in two-way trades. If users are given information about their trading partner, but not told now many edges are involved, a proposed pair-wise exchange would look exactly like an n-way exchange.

Nonetheless, the broker must give any information with care. In the above example, we cannot give $a$ so much information about $c$ that it identifies $c$. Because of the OID hint, $a$ knows where $c$ was when it received $a$'s OID. Revealing $c$'s identity to $a$ would violate *control*.

Other possible sources of information include using OID hints or information about a trading partner to link to other data sources, such as the web or histories of previous transactions, emails, and appointments. Human-subject studies are likely needed to understand how users will reason about exchanges and what information will allow them to make informed decisions. Previous studies sug-

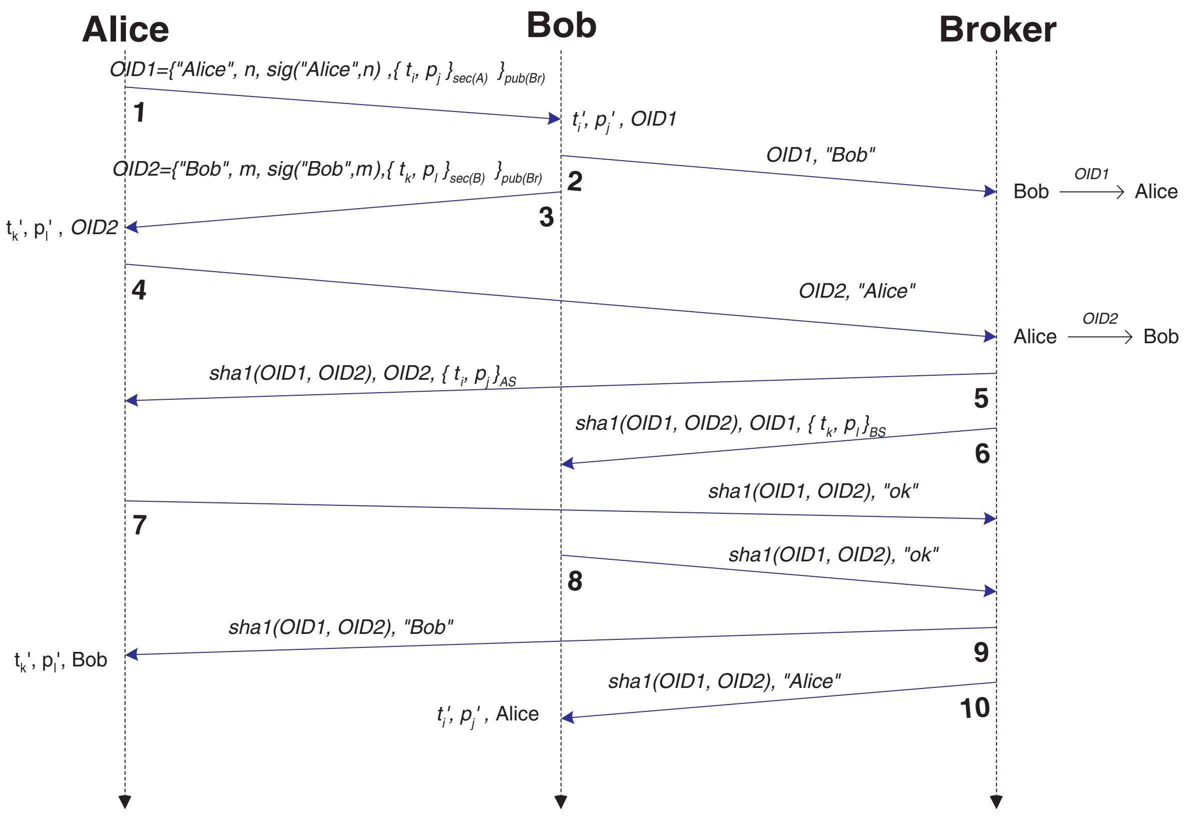

Alice
Bob
Broker
OID1={"Alice", n, sig("Alice",n) ,{ t_i, p_j }_sec(A) }_pub(Br)
1
t_i', p_j' , OID1
OID1, "Bob"
Bob –OID1→ Alice
OID2={"Bob", m, sig("Bob",m),{ t_k, p_l }_sec(B) }_pub(Br)
2
3
t_k', p_l' , OID2
4
OID2, "Alice"
Alice –OID2→ Bob
sha1(OID1, OID2), OID2, { t_i, p_j }_AS
5
sha1(OID1, OID2), OID1, { t_k, p_l }_BS
6
sha1(OID1, OID2), "ok"
7
sha1(OID1, OID2), "ok"
8
sha1(OID1, OID2), "Bob"
t_k', p_l', Bob
9
sha1(OID1, OID2), "Alice"
t_i', p_j' , Alice
10

**Figure 5: Example exchange**

gest that the identity of the requester is by far the most important piece of information. [6]. We intend to explore these issues further in our future work.

Finally, though hints provide dispersion, the lack of trusted authority also imposes limitations on the guarantees provided by SmokeScreen. One limitation is that while hints remind a user when they first broadcast an OID, this is not necessarily the same time and place when the OID was actually received by their exchange partner. This distinction can be exploited by an attacker to convince two people that they were in the same place when if they were not.

To see why, consider a scenario involving an attacker, $E$, and two geographically separated users, $A$ and $B$. $E$ can snoop on $A$ and $B$ as well as broadcast messages to them. If $A$ broadcasts $OID_a$ and $B$ broadcasts $OID_b$, $A$ will not receive $OID_b$ from $B$ and $B$ will not receive $OID_a$ from $A$. However, since $E$ can snoop on and broadcast to $A$ and $B$, it can receive both $OID_a$ and $OID_b$ and then replay those messages such that $A$ receives $OID_b$ and $B$ receives $OID_a$. If $A$ and $B$ then exchange resolutions, they will be under the false impression that they were co-located.

This attack can be detected if $A$ and $B$ are able to compare their respective OID hints; the location in $A$'s would be different than the location in $B$'s. Thus, to remove some of the uncertainty over where an interested party received an OID, the broker could act as a communication channel through which users are able to negotiate secret keys with one another without revealing their identities to each other. Under the protection of such a key, users could then exchange information about when and where they broadcast a particular OID without divulging location information to the broker.

## 5. IMPLEMENTATION

Our prototype system consists of three components: the mobile device, the broker, and the client.

### 5.1 Mobile Device

We have implemented our privacy controls on two devices: a BlueTooth-enabled Nokia 6670 mobile phone running Symbian OS and a Sharp Zaurus SL-5600 PDA with a Pretec CompactFlash type II 802.11b WiFi card running Linux 2.4.18.

For the 6670, our implementation mostly consists of 300 lines of Python run under Nokia's Python for Series60. Although writing applications in Python does not provide optimal performance it allowed us to quickly create a working prototype. The only non-Python code is a small addition to HIIT's Miso library [35] that allows Python code to set the BlueTooth device name. All RSA and AES primitives were implemented in pure Python via the PyCrypto library.

A primary concern for our mobile device was the power consumed by communication and cryptography. For the 6670, a daemon periodically wakes up to set the BlueTooth device name to an OID and four signals. Because the 6670 does not contain a GPS receiver, we do not currently include the device's location in OID hints. To reduce the amount of cryptography involved with computing an OID, user signatures are pre-computed. We also allow up to 48 hours of clique signals to be pre-computed.

To stay within the maximum BlueTooth device name size of 256-bytes, we used 128-bit AES symmetric and 1024-bit RSA asymmetric keys. After setting the BlueTooth name, the daemon executes BlueTooth "device-discovery" and logs the time and any devices it finds during the scan to flash storage. Importantly, the phone can respond to discovery inquiries from other BlueTooth devices while the daemon sleeps.

To compare presence-sharing using BlueTooth and WiFi, we also implemented our controls on the Zaurus PDA using the 802.11 card in "ad-hoc" mode to broadcast and receive OIDs. This code is written in C++ and uses the openssl library's RSA and AES primitives.

The daemon running on the Zaurus consists of two threads. A broadcasting thread wakes up periodically to compute and broadcast signals and OIDs. If the signals have been pre-computed and stored, the system uses them, otherwise it computes signals on the fly. It broadcasts all signals and the OID through the wireless card and goes back to sleep.

A receiving thread uses blocking I/O to scan for and receive clique signals and OIDs. The receiver distinguishes between the two using a 1 byte header. In the absence of transmission errors, it is easy to distinguish the two because OIDs are 128 bytes long while signal-index pairs are 32 bytes. The thread then checks to see if any of the clique signals are known. If it finds a matching clique signal, the broadcaster can be identified by decrypting the rest of the broadcast with the proper secret key.

## 5.2 Broker and Client

The broker and client are written in Java with a combined source tree of approximately 30 files and 4,000 lines of code. Both the broker and client are composed of a single long-lived server daemon, many small utility programs, and a PostgreSQL 7.4.7 database storing any persistent state. The daemons only receive network messages: the broker daemon registers interest in OIDs and receives votes; the client daemon receives trade terms and vote results. Messages are sent by several small utility programs: adduser, checkpending, finalizevotes, retrypending, searchgraph, and updateuser for the broker; setinterest and votetrades for the client.

These utility programs are triggered by the daemons via the dnotify utility, which runs event handler programs whenever specified file system events occur. For example, once trade terms have been safely inserted into the client's `trades` table, the client daemon reads from a file that has been registered with dnotify to run votetrades whenever it is read. Similarly, after the broker daemon registers interest in an OID it appends the identity of the interested user to a file registered with dnotify to run searchgraph whenever it is written.

As with our Python implementation of the mobile device, this is likely not the most performant way to implement the client and broker. For example, it would be more efficient to combine all functionality into a single long-lived process. However, distributing functionality between many programs accessing a common persistent store allowed us to quickly build and evaluate our prototype.

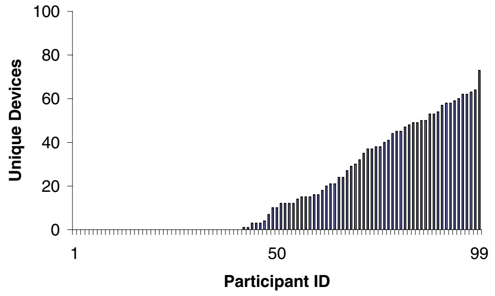
## 6. EVALUATION

In evaluating SmokeScreen's privacy controls, we set out to answer the following questions:

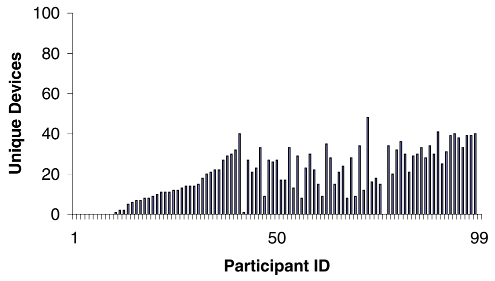
- Is there evidence of anonymous snooping in practice?
- How often can mobile devices broadcast signals and OIDs?
- Is the time to resolve an OID reasonable?

## 6.1 Reality Mining Analysis

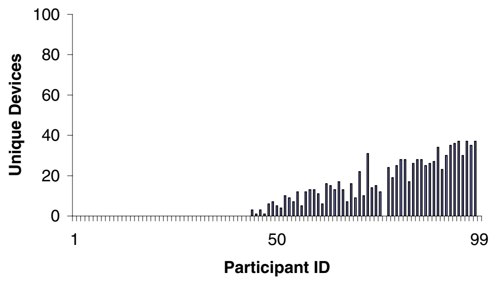
To look for evidence of anonymous snooping, we analyzed four months of mobility traces from the Reality Mining [9] project, from January 2005 through April 2005. The Reality Mining study logged co-presence information of 100 students and researchers from MIT over six months. Study participants were given Nokia 6600 mobile phones that ran BlueTooth device discovery and logging software after startup in the background. After each scan, the devices logged the time, discovered BlueTooth MAC addresses and device names,



(a) $|L|$



(b) $|M|$



(c) $|L \cap M|$

**Figure 6: Known devices**

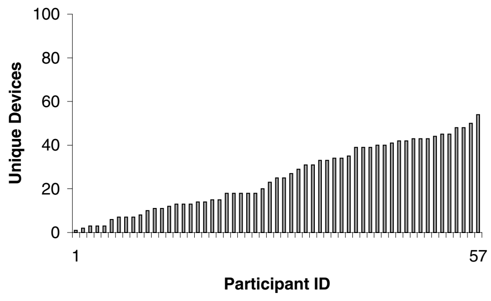
and cell tower information. Due to power constraints, the phones ran device discovery every five minutes.

We were most interested in two aspects of these traces: the set $L_i$ of devices that participant $i$ logged and the set $M_i$ of devices that logged participant $i$ over the course of four months. Because not all of the devices in the trace were known, we initially pruned trace entries to only include the 100 participants.

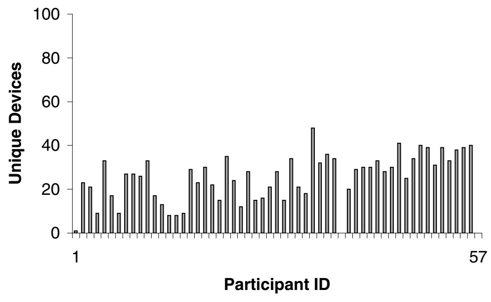
For our figures, identifiers were assigned by sorting participants in ascending order based on the size of $L$. Thus, participant 100 logged the most unique known devices and participant 1 the fewest. Figure 6(a) shows the size of $L$ for each participant and Figure 6(b) shows the size of $M$. There appears to be little correlation between the two.

Figure 6(c) shows the intersection of $L$ and $M$ for each participant. This gives us a sense of how symmetric the presence-sharing network was, i.e. whether when $a$ logged $b$, $b$ also logged $a$. In a perfectly symmetric network with no anonymous snooping, all three figures would be the same, but this is clearly not the case.

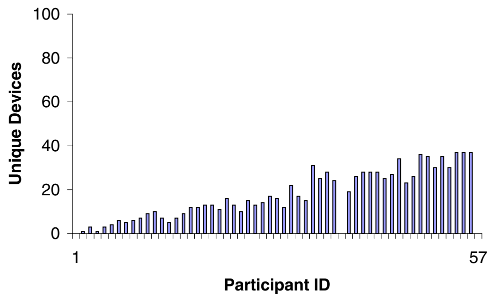
These figures show that many participants logged more devices than logged them and that others logged far fewer devices than logged them. For example, the participant with the largest $L$ was not logged by anyone else. Furthermore, 43 participants never logged anybody at all, though they were often logged by others.



(a) $|L|$



(b) $|M|$



(c) $|L \cap M|$

**Figure 7: Known loggers**

We considered three possible sources of this asymmetry. First, participants may have stopped logging in an attempt to conserve power. This would explain the 43 users who never logged anything, but were often logged by others. Figure 7(a), Figure 7(b), and Figure 7(c) show $L$, $M$, and $L \cap M$ as before, but with all non-logging users removed. Despite the removal of these users, asymmetry remains.

The asymmetry may also arise from users scanning every five minutes. Pairs of devices that are "out of phase" and near each other only long enough for one to log the other will lead to asymmetry. While this likely happened in individual interactions, over four months users would, on average, be the "logged device" half the time and the "hidden device" half the time. Thus, we would expect the aggregate effect of phase differences to be near zero.

To see if this was the case, we could not rely on the intersection of $L$ and $M$, since many interactions may have been one-time events. However, the *number* of times a participant was logged versus hidden does not distinguish between multiple interactions with the same user and several interactions with different users. Figure 8 shows the difference between the size of $L$ and the size of $M$ for known loggers, sorted from greatest to least. Note that the participant identifiers in Figure 8 are different from earlier figures.

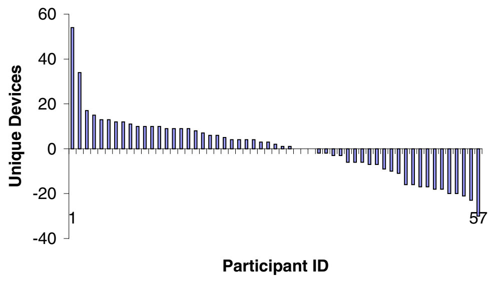


**Figure 8: Known loggers:** $|L| - |M|$

A negative value indicates that a participant was logged by more devices than it logged.

Again, we see that many of the busiest loggers were nearly hidden from other participants. This suggests another explanation: some users may have set their device to be non-discoverable. This behavior constitutes a form of anonymous snooping and violates our goal of *disclosure*. This is also quite surprising since study participants were members of a small community of mutually trusting, presumably altruistic researchers.

Importantly, while these findings are interesting, they do not allow us to make any definitive claims about users' intent; the asymmetry we observe is not necessarily a result of anonymous snooping. However, even if the asymmetry is due to out of phase devices or communication interference, at the very least our analysis demonstrates that anonymous snooping is feasible.

## 6.2 Power Consumption

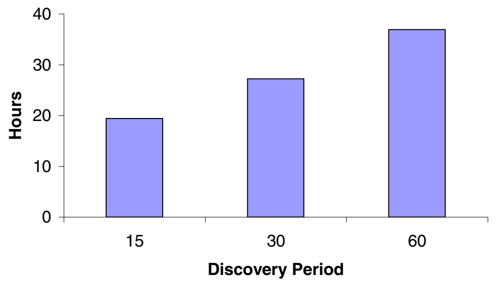
The next question we wanted to answer was whether the power demands running SmokeScreen on mobile devices are reasonable. To answer this, we ran experiments on both prototype devices. In each set of experiments, pairs of devices—either two Nokia 6670 mobile phones or two Sharp Zaurus PDAs—were fully charged, placed near each other in a sparse network environment, and then broadcast and logged signals and OIDs until their batteries ran out. For each experiment, we set the discovery period to 15, 30, or 60 seconds. In the mobile phone experiments, all signals were precomputed. In both cases, OIDs were computed at broadcast-time and discovery periods were set lower than one might expect in practice. Because of this, these results are overly conservative.

The results of the mobile phone experiments are in Figure 9(a) and of the PDA experiments in Figure 9 (b). The numbers are an average of four lifetimes with standard deviations less than one percent of the total.

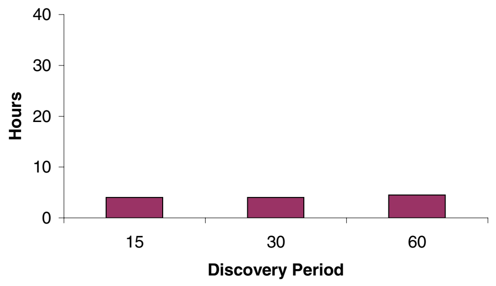
These figures show why BlueTooth is such an attractive platform for presence-sharing. The mobile phone scanned every minute and lasted over 36 hours on a single charge. The PDA performed significantly worse, lasting close to four hours regardless of the discovery period. The discovery period's lack of impact on the PDA's lifetime is due to never powering off the network card. We had to keep the network card on continuously so that devices were ready to receive messages when they were broadcast. BlueTooth handles this much more elegantly, dropping into a low power mode to wait for discovery requests. These results show that running our controls on a mobile phone is feasible.

## 6.3 Resolution Performance

The final issue we wanted to address is how long it takes to resolve an OID. To do this, we measured how long it took to resolve an OID from the moment it was first inserted into the client's presence table to the moment the broadcaster's identity was added.



(a) Mobile phone



(b) PDA

**Figure 9: Battery lifetime**

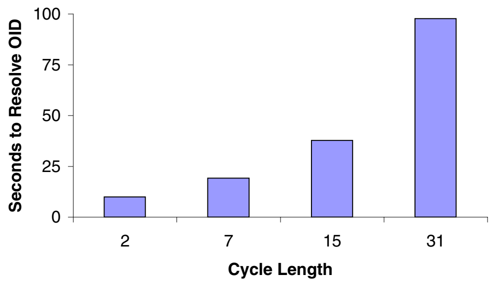


**Figure 10: Time to resolve OIDs**

We did this for several cycle sizes. For these experiments, exchange participants were the only users involved and we assumed that users voted favorably and immediately.

The broker and each client ran in a separate Xen virtual machine [2] with 256MB of memory on top of IBM x335 servers interconnected by 1GBps Ethernet. Each experiment ran 10 times. The results exhibited standard deviations of less than one percent and are in Figure 10.

These figures show that OIDs involved in large exchanges take much longer than those in small exchanges. Resolving an OID as part of a two-user exchange took 10 seconds while resolving an OID as part of a 31-user exchange took nearly 100 seconds.

We present these numbers merely as a *lower bound* on resolution performance. They do not include communication between the mobile device and client or the time users might need to decide how to vote. All of these factors will undoubtedly add to the expected time to resolve an OID. However, for applications such as mobile social networks and missed connection messaging, a delay of tens of seconds per resolution is reasonable.

## 7. RELATED WORK

One of the most interesting uses of presence information is the MMM [38] project, which associates environmental information with images. A later incarnation, called MMM2 [8, 30], uses co-presence to suggest other users with whom to share an image. Location privacy was not addressed by either project.

In addition to photo sharing, researchers have begun using presence-sharing to build other applications such as mobile social networks. One such system is Social Serendipity [10], which is part of the Reality Mining [9] project. Serendipity shares some architectural features with our mechanisms, but there are two key differences.

First, because Serendipity assigns static identifiers to users, obtaining a profile once allows users to resolve the profile owner's identity forward and backward in time in violation of *isolation*. OIDs and clique signals prevents this through timestamps and rotating secret keys. Second, Serendipity requires a trusted "BlueDar" server to be co-located with users. While reasonable for some social settings, we have tried to preserve the openness of our mechanisms. Thus, clique members can exchange presence information without a trusted intermediary. Also, though our broker is centralized and trusted, it serves without being co-located with users.

Privacy in ubiquitous computing is well-studied [19] [20] and much of this work guided the design goals in Section 4.

Gruteser and Grunwald [17] show a way to anonymize location data by means of a central server to prevent an untrusted server from from leaking sensitive information. This work assumes that the server is interposed between all user interactions, but presence-sharing presents a more challenging environment where users communicate with one another directly.

Another interesting paper from Gruteser and Grunwald [18] deals with the location privacy of WiFi users. To prevent surreptitious tracking of MAC addresses, they suggest "disposable" interface identifiers. This work is complementary to our own and as previously mentioned, we look forward to applying their ideas to BlueTooth.

Another important concern in location privacy is the management burden placed on users[29]. We believe that by grouping users into cliques, activation and deactivation of cliques will be easy to manage. Users could automate activation and deactivation depending on the time of day and their position. For example, the co-worker clique might automatically be activated at nine in the morning each week day and deactivated promptly at five the following afternoon.

Finally, Anagnostakis and Greenwald [1] have utilized cyclic demand in the context of file-sharing. Besides differences in target resource, the main difference between this work and ours is the location of the demand graph. While the broker is responsible for our demand graph, individual file sharers maintain their own graphs so that they can prioritize file request queues. This approach is inappropriate for presence-sharing, where identities must be hidden until resources are consumed.

## 8. CONCLUSIONS

We have designed and evaluated SmokeScreen's flexible, power-efficient controls for location privacy in presence-sharing networks based on the goals of control, disclosure, isolation, and dispersion. Sharing presence requires two complementary mechanisms: between social relations, such as a friend or a family member, privacy can be regulated using low-cost clique signals; between strangers, sharing requires opaque identifiers (OIDs), which can only be resolved to an actual identity by a trusted broker.

## 9. REFERENCES

[1] K. G. Anagnostakis and M. B. Greenwald. Exchange-based incentive mechanisms for peer-to-peer file sharing. In *Proceedings of the 24th International Conference on Distributed Computing Systems*, Tokyo, Japan, March 2004.

[2] P. Barham, B. Dragovic, K. Faser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, October 2003.

[3] J. Begole, J. Tang, R. Smith, and N. Yankelovich. Work rhythms: Analyzing visualizations of awareness histories of distributed groups. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work*, New Orleans, LA, November 2002.

[4] Cnn.com - MySpace's the place online, February 2006.

[5] P. Chown. Advanced encryption standard (MD5) ciphersuites for transport layer security (TLS). Internet RFC 3268, June 2002.

[6] S. Consolvo, Ian Smith, T. Matthews, A. Lamarca, J. Tabert, and P. Powledge. Location disclosure to social relations: Why, when, and what people want to share. In *CHI '05*, Portland, OR, April 2005.

[7] L. Cox, V. Marupadi, and A. Dalton. Presence-exchanges: Toward sustainable presence-sharing. In *Proceedings of the 7th IEEE Workshop on Mobile Computing Systems and Applications*, Semiahmoo Resort, WA, April 2006.

[8] M. Davis, N. Van House, J. Towle, S. King, S. Ahern, C. Burgener, D. Perkel, M. Finn, V. Viswanathan, and M. Rothenberg. MMM2: Mobile media metadata for media sharing. In *Extended Abstracts of the Conference on Human Factors in Computing Systems*, Portland, OR, April 2005.

[9] N. Eagle and A. Pentland. Reality Mining: Sensing complex social systems. *Journal of Personal and Ubiquitous Computing*, 2005.

[10] N. Eagle and A. Pentland. Social serendipity: Mobilizing social software. *IEEE Pervasive Computing*, pages 28–34, April–June 2005.

[11] facebook.
http://www.facebook.com/.

[12] M. J. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *Proceedings of the 23rd Annual Eurocrypt Conference*, Zurich, Switzerland, May 2004.

[13] Google Desktop.
http://desktop.google.com.

[14] S. Garriss, M. Kaminsky, M. J. Freedman, B. Karp, D. Maziéres, and H. Yu. RE: Reliable email. In *Proceedings of the 3rd Symposium on Networked Systems Design and Implementation*, San Jose, CA, May 2006.

[15] J. Gray. Notes on database operating systems. In *Operating Systems: An Advanced Course*, pages 394–481. Berlin: Springer-Verlag, 1978.

[16] R. Gross and A. Acquisti. Information revelation and privacy in online social network (the facebook case). In *Proceedings of the Workshop on Privacy in the Electronic Society*, 2005.

[17] M. Gruteser and D. Grunwald. Anonymous usage of location-based services through spatial and temporal cloaking. In *Proceedings of the 1st International Conference on Mobile Systems, Applications, and Services*, San Francisco, CA, May 2003.

[18] M. Gruteser and D. Grunwald. Enhancing location privacy in wireless lan through disposable interface identifiers: A quantitative analysis. *ACM Mobile Networks and Applications*, 10(3):315–325, June 2005.

[19] J. Hong and J. Landay. An architecture for privacy-sensitive ubiquitous computing. In *Proceedings of the 2nd International Conference on Mobile Systems, Applications, and Services*, Boston, MA, June 2004.

[20] J. Hong, J. Ng, S. Lederer, and J. Landay. Privacy risk models for designing privacy-sensitive ubiquitous computing systems. In *Designing Interactive Systems*, Cambridge, MA, August 2004.

[21] T. Kindberg, M. Spasojevic, R. Fleck, and A. Sellen. The ubiquitous camera: An in-depth study of camera phone use. *IEEE Pervasive Computing*, 4(2):42–50, 2005.

[22] M. Kochen, editor. *The Small World*. Ablex, 1989.

[23] A. LaMarca, Y. Chawathe, S. Consolvo, J. Hightower, I. Smith, J. Scott, T. Sohn, J. Howard, J. Hughes, F. Potter, J. Tabert, P. Powledge, G. Borriello, and B. Schilit. Place lab: Device positioning using radio beacons in the wild. In *Pervasive '05*, 2005.

[24] S. Lederer, J. Mankoff, and A. Dey. Who wants to know what when? privacy preference determinants in ubiquitous computing. In *Extended Abstracts of the Conference on Human Factors in Computing Systems*, Fort Lauderdale, FL, April 2003.

[25] macchanger.
http://www.alobbs.com/macchanger/.

[26] myspace.
http://www.myspace.com/.

[27] W. March and C. Fleuriot. The worst technolgoy for girls. In *Ethnographic Praxis in Industry Conference*, Redmond, WA, November 2005.

[28] R.J.T. Morris and B.J. Truskowski. The evolution of storage systems. *IBM Systems Journal*, 42(2):205–217, 2003.

[29] G. Myles, A. Friday, and N. Davies. Preserving privacy in environments with location-based applications. *IEEE Pervasive Computing*, 2(1):56–64, 2003.

[30] R. Nair and M. Davis. Bluetooth pooling to enrich co-presence information. In *Adjunct Proceedings of the 7th International Conference on Ubiquitous Computing*, Tokyo, Japan, September 2005.

[31] National Institute of Standards and Technology. Digital Signature Standard (DSS). FIPS Publication #186-2, January 2000.

[32] National Institute of Standards and Technology. Secure hash standard. FIPS Publication #180-1, April 1997.

[33] L. Palen and P. Dourish. Unpacking "Privacy" for a networked world. In *Proceedings of the Conference on Human Factors in Computing Systems*, Fort Lauderdale, FL, April 2003.

[34] S.N. Patel and G.D. Abowd. The ContextCam: Automated point of capture video annotation. In *Proceedings of the 6th International Conference on Ubiquitous Computing*, Nottingham, UK, September 2004.

[35] Personal Distributed Information Store.
http://pdis.hiit.fi/pdis/.

[36] R. Rivest. The MD5 message-digest algorithm. Internet RFC 1321, April 1992.

[37] Spotlight for Mac OS X.
http://apple.com.

[38] R. Sarvas, E. Herrarte, A. Willhelm, and M. Davis. Metadata creation system for mobile images. In *Proceedings of the 2nd International Conference on Mobile Systems, Applications, and Services*, Boston, MA, June 2004.

[39] C.A.N. Soules and G. Ganger. Connections: Using context to enhance file search. In *Proceedings of the 20th ACM Symposium on Operating Systems Principles*, Brighton, UK, October 2005.