Tryst: The Case for Confidential Service Discovery

Jeffrey Pang
Carnegie Mellon University

Ben Greenstein
Intel Research Seattle

Damon McCoy
University of Colorado

Srinivasan Seshan
Carnegie Mellon University

David Wetherall
University of Washington, Intel Research Seattle

ABSTRACT

Local service discovery exposes sensitive information about identity, location, and relationships. We present empirical evidence that suggests the release of this information in wireless environments poses a real danger to our privacy. We then discuss the challenges in the design of Tryst, an architecture that enhances the confidentiality of existing service discovery mechanisms.

1 INTRODUCTION

Mobile wireless devices, such as Zunes, PSPs, and iPhones, operate in environments where even devices on the same local network may be adversarial and anyone within communication range can overhear conversations. Not surprisingly, privacy concerns are driving the deployment of security mechanisms such as 802.11i, which are designed to authenticate clients and conceal the content of messages. However, such mechanisms do not conceal identities, locations, and relationships when a client attempts to discover a service in close proximity, a process we call *local service discovery*.

Local service discovery, hereafter referred to simply as service discovery, is an essential bootstrapping component of many applications in ad hoc environments. 802.11, Bluetooth, and Zigbee devices have mechanisms to discover other devices with which to communicate or access points from which to obtain connectivity. Windows, Mac OS X, and Linux operating systems use service discovery protocols such as NetBIOS, UPnP SSDP, SLP, and Multicast DNS (mDNS) to discover other devices and services on the same local network. User-level applications such as iTunes and iChat use service discovery to find other instances of the same application on the local network.

The discovery of a service, whether it be a network, an application, or another device, involves the broadcast of unencrypted *announcements* by services or *probes* by clients searching for services. Hence, application-level services reveal these messages not only to services that they trust but also to everyone on the local network; link layer services additionally reveal them to everyone nearby since they are sent before security associations are established. To date, no serious effort has been made to conceal these messages because service discovery has primarily been used to find public infrastructure that has little need for privacy (*e.g.,* the "tmobile" 802.11 network) or services on trusted wired networks that provide physical security. In addition, there has been little examination of how damaging the information exposed by service discovery can be.

We argue that the privacy threat is under-appreciated. For example, clients that probe for the names of infrastructure networks may reveal damaging information (*e.g.,* "Hooters - Plano" and "Juvenile Detention Classroom"[1]) because network names can be correlated with online databases to show where users have been and what they have been doing. Previous work showed that probes and announcements can also be used to fingerprint and track users over time [20]. Similarly, service announcements enable eavesdroppers to inventory physical sites and profile devices for vulnerabilities.

Moreover, emerging wireless consumer electronics are changing the way that service discovery is used. Many client devices now offer services such as file and music sharing (*e.g.,* the Microsoft Zune), and, thus, are vulnerable to the same threats as infrastructure services. Similarly, wireless devices are now used for social interaction (*e.g.,* gaming) where communicating devices may be owned by different people. Thus concealing client and service identities becomes necessary to hide social relationships.

In this paper, we present empirical evidence that demonstrates the severity of these privacy threats. We then discuss the challenges in designing *Tryst*, an architecture that eliminates information exposure by adding confidentiality to existing service discovery protocols. We focus on two essential areas: First, we discuss several service discovery protocols that are confidential *and* authenticated, highlighting the trade-offs in message overhead, denial-of-service resistance, and support for different privacy requirements. Second, because confidential discovery involves cryptographic keys, we discuss potential mechanisms to support key establishment in a range of scenarios without degrading ease-of-use.

[1] Real networks locatable via WiGLE [25] and Google Maps.

Figure 1—The two service discovery mechanisms.

2 THE THREAT

In this section, we describe the two service discovery mechanisms used by applications in ad hoc environments. We show that current implementations of these mechanisms expose sensitive information because they are neither confidential nor authenticated. Finally, we present a set of requirements for confidential service discovery motivated by these information leaks.

2.1 Service Discovery Mechanisms

Service discovery is used for one primary reason: user convenience. Service discovery enables users to locate, browse, and connect to useful services automatically. It is used both to discover if known services are present (*e.g.*, your home network) and to discover if previously unknown, but potentially useful services are present (*e.g.*, an airport network). Thus, it helps users avoid cumbersome configuration, which has historically been required to set up networks.

Although powerful infrastructure-based service discovery mechanisms exist (*e.g.*, those in SSDS [10] and INS [4]), they are not used in ad hoc environments because these environments lack such infrastructure and may not even have online connectivity. Instead, applications generally use two simple mechanisms that rely only on the existence of a communication medium between clients and services (Figure 1). They are:

Announcement. A service may periodically broadcast its identity (as well as other properties) to others nearby. For example, in 802.11, an access point (AP) periodically announces the name of its network, called a Service Set Identifier (SSID), so that nearby clients can identify the AP and associate with it.

Probing. A client may search for a service by name or description explicitly by broadcasting probes. For example, in mDNS, a client that wishes to resolve a local domain name of a device will broadcast a DNS query for that name to all other devices on the local network (via IP multicast). If that device is present, it hears the query and responds with a DNS reply. Some protocols allow clients to broadcast a probe asking for *any* service, which will generate a response from all services that receive it.

Applications often use both these mechanisms. For example, 802.11 clients can listen for announcements or explicitly probe for SSIDs.

Protocol Name	Ports	Devices (%)
NetBIOS	137,138	530 (69.9)
mDNS	5353,5355	270 (35.6)
UPnP SSDP	1900	260 (34.3)
SLP	427	106 (14.0)
Microsoft Office v.X	2222	50 (6.6)
Internet Printing Protocol	631	8 (1.1)
TiVo Beacon Protocol	2190	8 (1.1)
Dantz Retrospect	497	8 (1.1)
LiveTribe SLP	8427	6 (0.8)
BakBone NetVault	20031	4 (0.5)
Other		31 (4.1)
Any Protocol		685 (90.4)

Table 1—Most common application-level service discovery protocols observed at OSDI 2006 and the number of devices using each.

2.2 Privacy Leaks

These mechanisms result in the disclosure of a service's identity as this information is broadcast and anyone can overhear it. Moreover, neither party is authenticated during discovery, so anyone can detect the presence of a service or masquerade as it.

We argue that the information exposed by these service discovery mechanisms raises real privacy concerns. Privacy risks result from the exposure of any of four classes of information:

Inventory. Consumer devices that announce their existence or respond to probes can reveal what a user is carrying or the contents of a building or vehicle. Thieves can exploit this information to target thefts—*e.g.*, [21] reports that mobile phone pirates break into cars with phones that announce their presence. Inventorying attacks on these devices can be easier to carry out than the ones involving RFID [13] because they can be performed from a much greater distance. In addition, hackers can discover application services though announcements and probes in order to launch targeted attacks—*e.g.*, a device that sends mDNS probes or announcements for Apple services could be targeted to exploit a recent buffer overflow vulnerability [6]. Thus, maintaining the privacy of services as well as clients is important.

In general, service names enable adversaries to profile devices and users. Table 1 shows the application level service discovery protocols observed in a trace of wireless traffic at OSDI 2006 [8] and the number of devices using each.[2] The vast majority of devices (90.4%) engage in application-level service discovery and thus leak some information.

Location. Immobile, infrastructure services sometimes

[2]We distinguish devices by MAC address and only count devices that sent at least one DHCP request. Protocols were identified by port numbers in broadcast and multicast packet headers. Note that NetBIOS, mDNS, UPnP SSDP, and SLP messages may specify particular services (*e.g.*, iTunes), but we could not identify them because the trace did not contain packet payloads.

desire privacy in order to hide their location. For example, the practice of disabling 802.11 beacons is common [14] even though this practice does not completely hide a network's name. There are several legitimate reasons to hide infrastructure services: First, non-public services (*e.g.*, home and enterprise networks) that are not addressable by unauthorized users are more difficult to attack. For example, an attacker can not cause malicious packets to be processed if they are never received by the target. Second, the names of multi-site services can reveal where an organization is located. For example, searching for the SSID "IR_Guest" in WiGLE [25], a database that maps SSIDs to the geographic coordinates where they were observed by war drivers, reveals the locations of all the Intel Research labs. Although the location of these labs is not private, this type of information could be used to find physical sites that were not meant to be public.

SSIDs leak some information simply because they are descriptive, human readable names. However, even if they were opaque, they would still be consistent and could be correlated with other information. For example, knowledge of the SSID at one site can be used to infer locations of other sites by observing that SSID elsewhere.

A service could reveal its existence only to authorized clients by authenticating probes and encrypting replies. However, this solution requires clients to reveal their presence through probes without any indication that the service is trustworthy. Unfortunately, probes additionally expose the following two classes of information.

History. Probing can expose which services a user has associated with in the past. These associations can often be correlated with other sources of information to reveal a user's location and relationship history. For example, consider the SSIDs that were observed in 802.11 probe requests in a wireless trace collected at SIGCOMM 2004 [22]. We correlated these SSIDs with the coordinates of networks in WiGLE and matched each to the closest city or town (using the coordinates of 3,302,984 population centers from U.S. Census [24] and GNS [19] data). We note that the resolution granularity can actually be much finer than the city level. For example, four home networks observed in the SIGCOMM trace for which we knew the real locations resolve to within a few hundred feet of the real coordinates.

These locations suggest where SIGCOMM users have been. 390 of the 455 devices probed for at least one SSID (excluding the SSIDs of the SIGCOMM networks). 107 devices probed for at least one SSID that resolved to exactly one city. Although we do not have the ground truth information to verify whether these are actually locations that users have visited, we note that this process correctly finds locations visited by the authors.[3] For example, the most likely location of Jeffrey Pang's SSIDs is Pittsburgh, PA, which is where he lives. These probes also reveal relationships (*e.g.*, that Jeffrey Pang is associated with CMU).

We note that if devices never probed for specific services, they would not be vulnerable to this particular attack. However, without descriptive probes, services would be *required* to either constantly beacon their existence or to respond to all wild-card probes and thus could not have any privacy.

Clients could also limit information exposure by probing only for those services that they believe are present. However, this process would require manual user intervention because devices can not know this information a priori. Users may not want to sacrifice the convenience of automated discovery; studies have shown that humans often are willing to trade off long-term privacy for such short-term benefits, even when such a trade off is irrational [3]. Therefore, privacy protection mechanisms must not degrade ease-of-use if they are to be adopted.

Identity. In order to prevent user tracking, researchers have recognized the need to eliminate unique identifiers such as MAC addresses [16, 17]. However, in previous work, we showed that names in probes and announcements can be used to fingerprint users [15, 20]. This is because, even if devices never emit unique identifiers, the set of services that they attempt to discover and the set that they announce may be unique. Hence, adversaries can identify users by monitoring for these fingerprints.

2.3 Design Requirements

Irrespective of the aforementioned information leaks, new service discovery mechanisms are unlikely to replace current ones unless they retain their two most essential properties:

Plug-and-Play Networking. Users have grown to expect service discovery to find useful services in wireless environments with little or no human intervention. Therefore, service discovery mechanisms should remain as transparent as possible, keeping in mind that ease-of-use is often in conflict with security.

Infrastructure Independence. Mechanisms should continue to function in ad hoc environments with nothing more than a broadcast communication medium between clients and services. Note that man-in-the-middle attacks are much easier to perform in such environments than in the Internet so privacy is harder to maintain.

In order to protect privacy in wireless environments, two additional properties are required:

[3] We identified our devices using techniques from [20].

Confidentiality. Both a client and a service may desire privacy during service discovery. In particular, neither should have to expose its presence to undesired parties. In addition, third parties should not be able to determine client/service relationships, nor should any two discovery messages reveal information that links them to the same senders or receivers.

Authenticity. Authenticating clients and services is essential to prevent information exposure in the face of active adversaries (*e.g.*, man-in-the-middle attacks). Moreover, a client should be authenticated not only before it can access a service, but before it can even *learn about the presence* of a service. Similarly, a client must verify the authenticity of any service that it discovers before offering its own identity. This constitutes a reversal of existing practice where authentication occurs strictly after discovery.

3 DESIGN CHALLENGES IN TRYST

Existing service discovery implementations process probes and announcements in many different ways, so it would be foolhardy to introduce a new architecture that attempts to replicate the union of their functionality. Instead, Tryst supplements existing service discovery mechanisms by providing *access control* for probes and announcements, ensuring that only authorized devices can determine a message's sender, recipients, and contents. Message processing itself (*i.e.*, what happens when a discovery message is received by an authorized device) is delegated by Tryst to applications.

In this section, we discuss the challenges in the design of this access control primitive. These challenges fall into two areas: First, Tryst's discovery *protocol* must support a range of privacy requirements, not impose substantial overhead, and be resilient to denial of service attacks. Second, since cryptographic keys are necessary for authenticated and confidential discovery, Tryst must perform *key establishment* in ways that support common device introduction scenarios without degrading ease-of-use. We address each of these areas in turn, highlighting the design contributions Tryst makes and the research challenges that remain.

3.1 An Example

Before discussing Tryst's access control primitive, we describe how an existing application—802.11 network discovery—could use it for confidential service discovery. Suppose Bob trusts the 802.11 networks "Home," "Work," and "School," and they trust him. To discover if one of these networks is available, Bob's client takes a standard 802.11 ProbeReq and sends it via Tryst's access control primitive: Send({Home,Work,School}, ProbeReq). If Bob's Home AP is present, it will receive the ProbeReq and authenticate that Bob sent it. However, it will not learn the identity of any of the other intended recipients ("Work" and "School"). The AP will respond with a ProbeResp via Send(Bob, ProbeResp). No device other than Home, Work, and School, which represent unique identities known to Bob, will be able to determine the source, destinations, or contents of the probe and only Bob will be able to determine the source, destination, or contents of the response. Furthermore, the discovery messages reveal no information that can link them to future messages from Bob or his Home AP. Finally, messages sent after discovery can be made confidential by encrypting them with a secret key and addressing them using temporary pseudonyms [17]; both these items can be exchanged in discovery messages.

3.2 Protocol Challenges

To satisfy the *confidentiality* and *authenticity* requirements, the protocol that implements the Send access control primitive discussed above needs to authenticate the sender in a confidential manner.

Public Key Protocol. Abadi and Fournet [1] present such a *private authentication* protocol that is sufficient to implement Send using public keys as identities. Nonetheless, there are two concerns with this protocol.

First, the size of each probe or announcement scales with the number of potential recipients. This overhead might be impractical for announcing services that have hundreds or thousands of authorized clients. However, clients are unlikely to use more than a few services of a single type and thus the overhead of probe-based discovery might be acceptable. For example, wireless traces show that 90% of users at OSDI 2006 probed for at most 12 unique 802.11 SSIDs. Similarly, due to the structure of social networks, the average user is unlikely to establish many direct relationships.

Second, in order to protect the identity of the intended recipients, discovery messages are not addressed. Therefore, this protocol requires receivers to attempt to decrypt *every* discovery message that they receive, whether it is intended for them or not. To lessen this burden, we can use an encryption technique [7] that protects the identity of recipients, yet enables receivers to check whether each message is intended for them in constant time (independent of the number of recipients). However, each decryption attempt still involves public key cryptography and thus can take several milliseconds to complete. This leaves receivers open to simple denial of service attacks and slows services down when many devices in an area perform discovery simultaneously.

Symmetric Key Protocol. To address this pitfall, we note that the most common use of discovery is to discover known services (*e.g.*, a home 802.11 AP). There-

fore, a client and service will likely have met before, either out-of-band (see Section 3.3) or using the public key protocol described above. At that time, they can negotiate a secret symmetric key for future meetings. We independently developed a symmetric key protocol similar to the one proposed by Cox *et al.* [9] for detecting the presence of friends. In contrast to the public key protocol, this protocol enables devices to discard messages not intended for them efficiently by periodically computing unlinkable addresses known only to a client and a service. Nonetheless, a discovery message's size still scales with the number of intended recipients.

Since we expect this protocol to be sufficient for discovery in the common case, devices can prioritize processing of these messages while using spare cycles to process public key protocol messages. Thus, clients can discover both services that they have associated with before (using the symmetric key protocol) and services for which they only know the identity (using the public key protocol).

Discussion. Atypical clients that want to discover the presence of hundreds or thousands of services confidentially will incur substantial message overhead with either protocol described above. To reduce this overhead in a presence sharing application, Cox *et al.* [9] propose sharing a single key with cliques of mutually trusting friends. However, this makes changing trust relationships difficult because it requires global agreement among the members of a clique. This is particularly difficult for mobile devices that are often offline.

One attractive option is a class of encryption protocols that enable a sender to broadcast a message that has size *sublinear* in the number of authorized recipients, but can still only be decrypted by those recipients [11, 18]. Both public and symmetric key variants exist. These protocols are made possible by the additional assumption that no more than m revoked receivers collude or no more than r receivers are ever revoked. However, these protocols require key state and message overhead that are both superlinear in either m or r, and therefore would only be more efficient when the number of services a client wants to discover is larger. Moreover, their current instantiations reveal the sender's identity and relationships (*e.g.*, because they include the list of revoked devices). It is an open problem whether they can be made private [7].

Alternatively, one could reduce the number of services a client attempts to discover by ruling out services that are unlikely to be present based on context and location (*e.g.*, using GPS). Nonetheless, it is important that using context does not inadvertently expose identity or relationships (*e.g.*, discovery in different contexts should not noticeably change the number of discovery messages sent, lest message volume be used as a fingerprint).

Finally, we note that mutual confidentiality is not always required. If services (or clients) give up their privacy, more efficient protocols are possible. For example, a service willing to reveal its identity can simply announce its presence in an authenticated manner. Since the announcement need not address recipients, message size is constant. JFKi [5], a protocol originally designed for IPSec key exchange, can be used to provide client confidentiality and is resilient to denial of service attacks.

3.3 Key Establishment Challenges

Tryst's access control primitive enables confidential and authenticated service discovery, but requires keys that identify trusted devices and services. In order to maintain the *plug-and-play* and *infrastructure independence* properties of existing service discovery mechanisms, Tryst must be able to obtain these keys automatically without relying on the presence of trusted third parties during the discovery process. In particular, clients should still be able to discover *previously unknown* services that they would nonetheless have reason to trust.

The two traditional classes of key establishment mechanisms, *pairing* and *certificates*, can be used in some scenarios, but we argue that they are insufficient to establish keys in many important confidential service discovery settings. Therefore, we describe two mechanisms in Tryst that enable automated, confidential discovery in several of these novel settings. The first mechanism enables devices to establish keys with new devices belonging to entities that they trust by using a common naming convention. The second enables devices to establish keys with any device transitively via a trusted friend who trusts it. We illustrate these mechanisms with two common discovery scenarios below.

Existing Mechanisms. The two traditional classes of key establishment mechanisms are pairing and certificates. Pairing[4] [23] refers to the techniques used to establish keys on two personal devices that a user wants to connect together (*e.g.*, Bluetooth peripherals). Most of these mechanisms assume that users of the devices can identify them physically, which is often not the case (*e.g.*, when trying to find an 802.11 AP). Moreover, all these mechanisms assume that a client *already knows* the specific service it wants to discover. This may be true in the examples that we have discussed so far, but service discovery is often useful because it enables users to find services they do not yet know about (two such scenarios are described below). Ironically, this assumption means that to use pairing, *users* have to discover services before their *devices* can discover them, which defeats the purpose of "automatic" service discovery.

Secure websites offer certificates signed by trusted authorities (*e.g.*, VeriSign) to prove their authenticity to

[4]Note that our use of the term *pairing* in this paper is unrelated to its use for describing particular bilinear maps.

clients. However, private services can not offer certificates to unknown clients without violating their own privacy. Similarly, clients can not privately offer proof of identity before authenticating a service. Even predistribution of certificates via out-of-band channels is difficult because mobile devices are often disconnected.

New Devices in Trusted Domains. Devices in two mutually trusted domains can often assume bilateral trust. For example, if Alice and Bob are friends, they may allow all their current and future devices to discover each other. They might naïvely try to achieve this either by sharing a single private key among all devices in one domain, or by exchanging all their device keys. However, the former approach compromises all the devices if even one is stolen, and the later approach does not enable the discovery of new devices that Alice and Bob obtain after the key exchange.

To establish keys automatically in this scenario, Tryst leverages *anonymous identity based encryption* [2] (AIBE), a public key encryption scheme primarily used for confidential email. Using AIBE, Alice can assign a different private key to each device and Bob can encrypt a discovery message to that device using a human-readable string as the encryption key and a publicly known encryption algorithm. This string is chosen based on an agreed upon naming convention (*e.g.*, Alice.iPhone can be the encryption key for Alice's iPhone) but a message encrypted with it hides both the key and the recipient. A trusted authority provides Alice with the private decryption key, ensuring that only she can decrypt messages encrypted with keys beginning with Alice (*i.e.*, a unique username she uses with this authority). The authority also publicly publishes the encryption algorithm, which is the same for all its users (*i.e.*, the same algorithm is used whether the key is Alice.iPhone, or Alice.Zune, or Charlie.iPhone, etc.).

For example, suppose Bob and Alice each purchase a new iPhone, each preloaded with AIBE private keys. Bob configures his iPhone to trust (the string) Alice and Alice configures hers to trust Bob (*e.g.*, by adding each string to their respective address books). If Bob and Alice are nearby, their iPhones could discover each other and use 802.11 to connect their calls, without first needing to exchange keys (indeed, Bob need not even know that Alice has an iPhone). To do this, Bob's iPhone simply sends a discovery message encrypted using the string Alice.iPhone and only Alice's iPhone can receive it.

One downside of AIBE is that messages encrypted for users of one trusted authority are distinguishable from messages encrypted for users of another. Therefore, a discovery message reveals the trusted authority the recipient uses. If a sufficient number of people use this trusted authority, then Alice and Bob may permit this small degradation of privacy.

New Transitively Trusted Devices. Two mutually trusted devices may also be willing to trust each other's social relations. For example, Bob's iPhone may permit all 802.11 APs that Alice's iPhone uses to discover it. Similarly, some of these APs may permit Alice's friends to discover them. Bob might naïvely attempt to bootstrap keys for these APs by having Alice give him all their public keys. However, this approach does not work for new APs that Alice uses after this key exchange, and it forces Alice to reveal all her AP relations to Bob.

Tryst leverages a private social proximity test [12] to automatically establish keys in this scenario. This test, which can be integrated with the symmetric key protocol described in Section 3.2, enables Bob's iPhone to send a message that can only be read by devices that Alice has allowed to trust her friends, without having to explicitly tell Bob about any of them. Note that Bob will still learn about Alice's relationship with an AP when he discovers it. However, Alice, who must agree to use this mechanism, may be willing to permit this revelation.

Discussion. Although neither of these mechanisms provide absolute confidentiality as they require additional trust assumptions, they are only used when devices attempt to discover others with which they have not established keys. The symmetric key protocol (see Section 3.2) can be used for subsequent discovery attempts. Thus, an important research challenge is to determine how to limit the use of these mechanisms to those circumstances for which they are truly needed. For example, in the case of 802.11, clients generally prefer known APs to unknown ones. Thus, these mechanisms are not needed when known APs are present.

Finally, the rapid evolution of service discovery scenarios means that we do not yet have a clear picture of all use cases. Emerging scenarios may require access based on more dynamic capabilities than these mechanisms enable. For example, a wireless network may want to be visible only to clients in a specific physical area. Key establishment mechanisms for these scenarios are an important area of future work.

4 Status and Future Work

This paper presents the privacy threat posed by service discovery protocols as well as the challenges to removing it, and thus takes the first step towards the design of more confidential protocols. We have implemented Tryst and integrated it with a real 802.11 driver with the hope that this prototype will evolve into a valuable tool for protecting privacy. Through the use of this prototype, we expect to provide a better understanding of how confidential service discovery is used in the wild and to yield insight into practical considerations for the design of such systems. We believe these findings will be essential to establishing best practices for future service discovery protocols.

ACKNOWLEDGMENTS

This work is supported by the Army Research Office through grant number DAAD19-02-1-0389. This work was partially done while Jeffrey Pang and Damon McCoy were employed by Intel. We thank Michael Buettner, Tadayoshi Kohno, Sergiu Nedevschi, Anmol Sheth, and the HotNets reviewers for their valuable comments and suggestions.

REFERENCES

[1] M. Abadi and C. Fournet. Private authentication. *Theor. Comput. Sci.*, 322(3):427–476, 2004.

[2] M. Abdalla, M. Bellare, D. Catalano, E. Kiltz, T. Kohno, T. Lange, J. MaloneLee, G. Neven, P. Paillier, and H. Shi. Searchable encryption revisited: Consistency properties, relation to anonymous IBE, and extensions. In *CRYPTO*, 2005.

[3] A. Acquisti and J. Grossklags. Privacy and rationality in individual decision making. *IEEE Security and Privacy*, 3(1):26–33, 2005.

[4] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley. The design and implementation of an intentional naming system. In *SOSP*, Dec. 1999.

[5] W. Aiello, S. M. Bellovin, M. Blaze, J. Ioannidis, O. Reingold, R. Canetti, and A. D. Keromytis. Efficient, DoS-resistant, secure key exchange for internet protocols. In *CCS*, 2002.

[6] Apple Mac OS X mDNSResponder buffer overflow vulnerability. US-CERT, May 2007. `https://www.kb.cert.org/vuls/id/221876`.

[7] A. Barth, D. Boneh, and B. Waters. Private encrypted content distribution using private broadcast encryption. In *Financial Crypto*, 2006.

[8] R. Chandra, R. Mahajan, V. Padmanabhan, and M. Zhang. CRAWDAD data set microsoft/osdi2006 (v. 2007-05-23). `http://crawdad.cs.dartmouth.edu`.

[9] L. P. Cox, A. Dalton, and V. Marupadi. Smokescreen: flexible privacy controls for presence-sharing. In *MobiSys*, 2007.

[10] S. E. Czerwinski, B. Y. Zhao, T. D. Hodes, A. D. Joseph, and R. H. Katz. An architecture for a secure service discovery service. In *MobiCom*, 1999.

[11] A. Fiat and M. Naor. Broadcast encryption. In *CRYPTO*, 1993.

[12] M. J. Freedman and A. Nicolosi. Efficient private techniques for verifying social proximity. In *IPTPS*, 2007.

[13] S. Garfinkel, A. Juels, and R. Pappu. RFID privacy: An overview of problems and proposed solutions. *IEEE Security and Privacy*, 3(3):34–43, May/June 2005.

[14] M. S. Gast. *802.11 Wireless Networks*, page 175. O'Reilly, 2nd edition, 2005.

[15] B. Greenstein, R. Gummadi, J. Pang, M. Y. Chen, T. Kohno, S. Seshan, and D. Wetherall. Can Ferris Bueller Still Have His Day Off? Protecting Privacy in an Era of Wireless Devices. In *HotOS XI*, 2007.

[16] M. Gruteser and D. Grunwald. Enhancing location privacy in wireless LAN through disposable interface identifiers: A quantitative analysis. *ACM MONET*, 10, 2005.

[17] T. Jiang, H. Wang, and Y.-C. Hu. Preserving location privacy in wireless LANs. In *MobiSys*, 2007.

[18] D. Naor, M. Naor, and J. Lotspiech. Revocation and tracing schemes for stateless receivers. In *CRYPTO*, 2001.

[19] NGA GEOnet names server, July 2007. `http://earth-info.nga.mil/gns/html/index.html`.

[20] J. Pang, B. Greenstein, R. Gummadi, S. Seshan, and D. Wetherall. 802.11 user fingerprinting. In *MobiCom*, Sept. 2007.

[21] Phone pirates in seek and steal mission. Cambridge Evening News, Aug. 2005. `http://www.cambridge-news.co.uk/news/region_wide/2005/08/17/06967453-8002-45f8-b520-66b9bed6f29f.lpf`.

[22] M. Rodrig, C. Reis, R. Mahajan, D. Wetherall, J. Zahorjan, and E. Lazowska. CRAWDAD data set uw/sigcomm2004 (v. 2006-10-17). `http://crawdad.cs.dartmouth.edu`.

[23] J. Suomalainen, J. Valkonen, and N. Asokan. Security associations in personal networks: A comparative analysis. Technical Report NRC-TR-2007-004, Nokia Research Center, Jan. 2007.

[24] U.S. census bureau - TIGER/line, 2000. `http://www.census.gov/geo/www/tiger/`.

[25] WiGLE: Wireless geographic logging engine. `http://www.wigle.net/`.