ABSTRACT

Title of dissertation: SHARING PRIVATE DATA
OVER PUBLIC NETWORKS

Randolph C. Baden, Doctor of Philosophy, 2012

Dissertation directed by: Professor Bobby Bhattacharjee
Department of Computer Science

Users share their sensitive personal data with each other through public services and applications provided by third parties. Users trust application providers with their private data since they want access to provided services. However, trusting third parties with private data can be risky: providers profit by sharing that data with others regardless of the user's desires and may fail to provide the security necessary to prevent data leaks. Though users may choose between service providers, in many cases no service providers provide the desired service without being granted access to user data. Users must make a choice: forego privacy or be denied service.

I demonstrate that fine-grained user privacy policies and rich services and applications are not irreconcilable. I provide technical solutions to privacy problems that protect user data using cryptography while still allowing services to operate on that data. I do this primarily through content-agnostic references to data items and user-controlled pseudonymity. I support two classes of social networking applications without trusting third parties with private data: applications which do not require

data contents to provide a service, and applications that deal with data where the only private information is the binding of the data to an identity. Together, these classes of applications encompass a broad range of social networking applications.

SHARING PRIVATE DATA OVER PUBLIC NETWORKS

by

Randolph C. Baden

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2012

Advisory Committee:
Professor Bobby Bhattacharjee, Chair/Advisor
Professor Ashok Agrawala
Professor Peter Keleher
Professor Mark Shayman, Dean's Representative
Professor Neil Spring

Dedication

This thesis is dedicated to my brother, Billy. Without his love, his support, and, perhaps most importantly, his bone marrow, I certainly could not have completed it.

Acknowledgments

There are many people who I would like to thank for a variety of reasons, without whom this thesis would not be possible. First and foremost, I would like to thank my advisor, Bobby Bhattacharjee. Bobby both taught me how to identify problems and gave me the tools I needed to tackle them. He has given me more opportunities to succeed than I can count, and without his guidance and understanding I would never have been able to produce the work contained in this thesis.

I would also like to thank the other professors at the University of Maryland, especially my committee members: Neil Spring, Peter Keleher, Mark Shayman, and Ashok Agrawala. Neil in particular has almost been like a second advisor to me, and has provided many invaluable lessons about research, measurement, and technical writing.

Likewise, I have learned a lot from the many students who I have had the honor of sharing a lab with over the past seven years: Rob Sherwood, Cristian Lumezanu, David Levin, Adam Bender, Aaron Schulman, Katrina LaCurts, Vassilis Lekakis, Kristin Stephens, Kevin McGehee, Hailey Lin, Kelly Lai, Jessy Kate-Schingler, Karla Saur, Abdul Quamar, Greg Benjamin, Yunus Basagalar, and Ramakrishna Padmanabhan. Without these people the long hours and sleepless nights would not have been bearable.

I would also like to thank my collaborators, some of whom I've already mentioned: Adam Bender and Daniel Starin for their work on Persona, Matt Lentz for his work on Twain, and Cristian Lumezanu and David Levin for allowing me to work

on (and learn from) Peerwise. I would also like to thank Alan Mislove for providing valuable comments and data for much of my work.

I would like to thank NSF and LTS for their generous contributions in the form of grants that have made this work financially possible. LTS has also provided valuable resources for feedback on much of my work that has directly led to better research practices. I would also like to thank UMD for giving me the opportunity to participate in its graduate program.

Finally, I would like to thank all of the people who provided non-technical support for my time in graduate school, including: my family; my friends; my doctors, Aaron Rapoport and Saul Yanovich; my nurses; and last but not least, my significant other, Edward Clifford.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

PKC	Public Key Cryptography
ABE	Attribute-Based Encryption
PKI	Public Key Infrastructure
OSN	Online Social Network
PGP	Pretty Good Privacy
AS	Autonomous System
$u.SS$	u's storage service location
$u.K$	Key K created by u
(PK, TSK)	PKC public/secret keypair
$(APK, AMSK)$	ABE public/master secret keypair
ASK	ABE user secret key
$\mathbb{AS}$	Access structure
$\mathsf{TKeyGen}()$	Generate RSA keypair
$\mathsf{TEncrypt}(K, m)$	RSA encrypt m with key K
$\mathsf{TDecrypt}(K, c)$	RSA decrypt ciphertext c
$\mathsf{TSign}(K, m)$	RSA sign m with key K
$\mathsf{ABESetup}$	Create attribute public key and master secret key
$\mathsf{ABEKeyGen}(K, attrs)$	Create attribute secret key with attributes $attrs$
$\mathsf{ABEEncrypt}(K, m, \mathbb{AS})$	ABE encrypt m with K and access structure $\mathbb{AS}$
$\mathsf{ABEDecrypt}(SK, PK, c)$	ABE decrypt ciphertext c with secret key SK
RS	Rendezvous service
$\hat{X}$	Pseudonym for user X
P_X	Properties of user X
$Q_{\hat{X}}$	Query for user X
(k, t, v)	Key, type, value triplet
$m_{\hat{X}}$	Message from user X, of the form $\{(k_i, t_i, v_i)\}$
$d_{\hat{X}}$	Thunk from user X to instruct further communication
$g(m_1, m_2)$	Comparison function for two collections of (k, t, v) tuples
$c(v_1, v_2)$	Comparison function for individual (k, t, v) tuples
$b(c, v)$	Bounding function for comparison function c and value v

Chapter 1

Introduction

The rise of online social networks (OSNs) and mobile devices in recent years has triggered a new era of *applications* that allow users to easily share information on the public Internet. These applications provide novel services that, judging by their popularity [100], are highly desired. In OSNs such as Facebook, Twitter, and Google+, users use these applications to share personal data such as gender, religion, and photos with friends. However, they also reveal that data to the service providers, sometimes deliberately and willingly, sometimes accidentally or unknowingly [106]. Mobile devices have compounded the problem, by providing fine-grained location information [110] and other mobile information [109] automatically to these applications – highly private information that concerns users who are not always sure how to protect themselves from its exposure.

Even if the user trusts a provider with her personal data, she must also trust that they will be able to protect it; several unintentional data leaks from otherwise trustworthy providers [115, 116] have been publicly revealed in recent years. Unlike in the case of leakage of, say, credit card information that leads directly to observable fraud, it is difficult to anticipate or detect the direct and indirect means by which a malicious entity might profit from personally identifiable information leaked in this way. Research has illuminated the consequences of some of these leaks [46, 56], sug-

gesting that it is a significant problem that sometimes may even take years to arise after the data is initially exposed. The true scope of this problem is immeasureable: not only can we not fully assess the damage from the leaks that are known, but we also cannot know if providers have had security breaches that they have not revealed to the public.

Nevertheless, users continue to trust providers with their private data because they want to use the applications that those third parties provide. Due to the nature of these services and their popularity, users are under social pressure to provide their data to these providers [35]. Unfortunately, it is the providers and not the users who get to decide what the rules are for sharing private data: if the user wants to use the provider's services, she must agree to their terms [105, 108] about her private data. Providers have little incentive other than potentially embarrassing public relations gaffes to protect user data, but they have plenty of economic incentive to share that data with advertisers or third party applications. Thus, current online social networks retain excessive control over user data and users must make a difficult choice between agreeing to the providers' terms or abandoning the service.

I argue the following thesis: it is possible to provide technical mechanisms to share private data over public networks, mechanisms that support desired user privacy policies while maintaining a wide range of application functionality.

I envision systems in which users retain control over the disclosure of their data, deciding precisely which other entities should be able to access which data items. There are many challenges to providing the rich application functionality to which users are accustomed while respecting the privacy policies that users may

choose. Though the challenges run the gamut from usability, to economic viability, to deployment, I focus primarily on providing solutions to the technical challenges associated with applications that handle private data. That is, I provide cryptographic and system-level tools that enable users to explicitly control to whom their private data is exposed. By providing this control locally, the user does not need to trust her service providers with unfettered access to her data, nor does she need to rely on her service providers to prevent data breaches.

To support my thesis, I first require an underlying framework for user-defined privacy that is compatible with how users wish to disclose data in OSNs. OSN communication takes many forms, including: one-to-one communication in the form of private messages, one-to-many communication in the form of user-specified groups such as "friends", and one-to-many communication to (perhaps unknown) recipients who are identified by others as in the case of friend-of-friend communication. I describe a secure OSN communication substrate as a foundation for building the case for my thesis. This supporting work is described primarily in Chapter 2 with some additional support in Chapter 3.

Based on my OSN communication framework, I consider the varied nature of OSN applications and identify large, generic classes of applications that I can reason about abstractly. I identify a class of applications that are *content-agnostic*, that is, applications that do not require access to data contents in order to provide service. I discuss these applications in Chapter 2, especially with regards to content-agnostic core OSN applications. The second class of applications that I identify are those which operate on *semi-private* data, i.e., data whose contents are not

inherently private, but the linkability of that data to an identity is considered private information. I describe many examples of these applications and provide a general framework for providing privacy in these applications in Chapter 4. The remaining class of applications fall outside of these two classes; such applications *require* access both to data contents and to the mapping from data contents to identities.

Finally, I must identify how these classes of application services can be provided while respecting the users' privacy policies. I describe how to do so for any content-agnostic application in Chapter 2. I consider many example applications of semi-private applications and possible user privacy policies that can be provided in Chapter 4, though in practice every application in this category requires application-specific analysis to determine which privacy policies can be provided. For the remaining class of leftover applications, I do not provide better solutions than that of trusting the third party. Further dividing this final class of applications to provide privacy in other ways is described in Chapter 6 as a topic of future work.

This dissertation is structured as follows. Chapter 2 describes Persona [6], a distributed and decentralized OSN in which users choose with which provider to store their data and protect their data content via encryption. Persona uses attribute-based encryption (ABE) to intuitively share data with groups of friends and even friends of friends. I describe how key OSN applications are implemented in Persona despite the encryption of data contents.

Chapter 3 describes Bond Breaker [7]. Bond Breaker is a demonstration that users can thwart OSN impersonation – a significant threat to the key distribution problem in decentralized OSNs – using exclusive shared knowledge. It also demon-

strates that users can collaboratively create a decentralized OSN PKI in-band, a valuable component for OSNs where users may not be willing to go out of their way for security.

In Chapter 4 I describe Twain, a pseudonymous rendezvous abstraction that can be used as a building block for privacy-enabled systems. Twain allows two users to rendezvous subject to the constraints of both users, so that users only reveal their true identities to each other if they are satisfied that the other user can be trusted. Twain provides a framework to implement user privacy policies for applications that require some amount of matching on private data contents.

Finally, in Chapter 6 I discuss the implications of this work. I describe open problems in this space, both in terms of technical challenges that are immediately relevant to this work and in terms of non-technical challenges that nevertheless impede the deployability of a new, privacy-enabled infrastructure. I conclude with an argument that my work describes the right approach to solving the complex problems of general privacy-enabled applications.

Chapter 2

Persona

2.1 Introduction

OSNs have become a de facto portal for Internet access for millions of users. These networks help users share information with their friends. Along the way, however, users entrust the social network provider with such personal information as sexual preferences, political and religious views, phone numbers, occupations, identities of friends, and photographs. Although sites offer privacy controls that let users restrict how their data is viewed by other users, sites provide insufficient controls to restrict data sharing with corporate affiliates or application developers.

Not only are there few controls to limit information disclosure, acceptable use policies require both that users provide accurate information and that users grant the provider the right to sell that information to others. Facebook is a representative example of a social network provider.

Cryptography is the natural tool for protecting privacy in a distributed setting, but obvious cryptographic schemes do not allow users to scalably define their privacy settings in OSNs. Users want to be able to share content with entire groups, such as their friends, their family, or their classmates. Public key cryptography alone is unsatisfactory when managing groups in an OSN: either users must store many copies of encrypted data, users are unable to give data based on membership in

multiple groups, or users must know the identities of everyone to whom they give access.

To meet the privacy needs of an OSN, we[1] present Persona, an OSN that puts policy decisions in the hands of the users. Persona uses decentralized, persistent storage so that user data remains available in the system and so that users may choose with whom they store their information. We build Persona using cryptographic primitives that include attribute-based encryption (ABE), traditional public key cryptography (PKC), and automated key management mechanisms to translate between the two cryptosystems.

Persona achieves privacy by encrypting private content and prevents misuse of a user's applications through authentication. Persona allows users to store private data persistently with intermediaries, but does not require that users trust those intermediaries to keep private data secret. Modern web browsers can support the cryptographic operations needed to automatically encrypt and decrypt private data in Persona with plugins that intercept web pages to replace encrypted contents. Lastly, Persona divides the OSN entities into two categories: users, who generate the content in the OSN, and applications, which provide services to users and manipulate the OSN content.

This chapter is organized as follows. We describe the cryptographic primitives and how they comprise the correct cryptographic systems for Persona in Section 2.2. We present novel compositions of ABE and PKC functions that allow users to create

[1]This work [6] involved the collaboration of Adam Bender, Bobby Bhattacharjee, Neil Spring, and Daniel Starin.

flexible and dynamic access policies in Section 2.3. We describe the role of OSN applications in Persona and show that Persona supports existing OSN applications in Section 2.4. We present significant features of our implementation in Section 2.5. We evaluate the performance of Persona using data from a Facebook crawl and ABE microbenchmarks on a mobile device in Section 2.6. We discuss additional problems beyond the scope of this work in Section 2.7, and conclude with some retrospection about how OSNs have changed between when we conducted this research and the present day in Section 2.8.

2.2 Cryptography in a Private OSN

There are two tasks for encryption in building the private online social network. The first is to restrict the information available to applications as precisely as possible, so that individual organizations are not entrusted with large volumes of personal information. Although it is tempting to focus only on the exchange of information with friends, some applications may benefit from limited access to a user's profile, location, or messages, while carefully avoiding broad exposure.

The second task is to restrict the information shared with "friends" to what might be appropriate. We quote "friends" here because the type of social link might be more than, less than, or different from "friend." Family, neighbor, co-worker, boss, teammate, and other relations might define a connection in the social network. That connection is often simply termed "friend", regardless of the actual, off-line relationship. A user's decision to accept one of these pseudo-friends into their neighborhood

(and avoid discussing certain topics) or exclude them (and avoid the benefits of social networking) represents a dilemma that can be avoided, if users may flexibly classify their "friends."

Alone, these two problems may be easily solved. A social network could help users define access policies that include or exclude defined groups of friends accessing different pieces of information. Such a feature would allow a user to tweet "called in sick to work" without telling co-workers. In practice, users segregate work colleagues from personal friends by subscribing to different social networks. To provide such functionality efficiently without the assistance of a trusted application provider requires some form of cryptographic support for group keying. In this section, we define two methods to share information with groups in an OSN.

What makes the OSN setting different from typical group keying scenarios is that the sender (to the group) may not be in charge of group membership. For example, Alice may post a message on Bob's wall, encrypted for Bob's friends, without (necessarily) knowing the list of Bob's friends. Further, Alice might wish to send a message to Bob's friends who live in the neighborhood: "Let's meet up tonight". Another aspect of the OSN setting is that the number of potential groups a user might encrypt to is very large (any possible combination of friends of their friends). Cryptographic support alone is not sufficient for building a distributed online social network; it is merely a necessary tool, difficult to apply, which shapes the eventual design.

2.2.1 Model

With the abstract goals of hiding personal information from aggregators and hiding personal information from colleagues, we next refine these goals down to concrete requirements for cryptographic methods.

Each Persona user generates an asymmetric key-pair and distributes the public key out-of-band to other users with whom she wants to share data. We refer to these other users as friends, though the nature of each relationship is defined by the user.

Persona allows users to create "groups" and choose which users are part of a given group. Users control access to personal data by encrypting to "groups." Restricting data to specific groups allows users to have fine-grained control over access policy, which permits exchanging data with more restrictions.

Cryptographic primitives in Persona must allow users to flexibly specify and encrypt to groups. Users may specify groups using arbitrary criteria, but we expect users to choose groups based on transparent relationships such as "neighbor" or "co-worker" or on attributes such as "football fan" or "knitting buddy." Groups created by one user do not affect the groups that can be created by another. However, to support OSN communication patterns, the groups created by one user should be available for use, not just for decryption, but also for encryption, by friends.

2.2.2 Traditional public-key approach

Traditional public-key and symmetric cryptography can be combined to form an efficient group encryption primitive [62, 92]. To create a new group from a list

of known friends, Alice encrypts a newly-generated group key with the public key of each member of the new group. She then distributes this key to the members of that group and uses the key to encrypt messages to the group. The group key may be symmetric, in which case only group members can encrypt to the group, or asymmetric, which allows non-members to encrypt as well.

Distributing a new group key may coincide with sending a new message: to create a message for all of her friends, Alice might include both the keys and the data in the same object for efficiency. To efficiently reuse a group and key for many messages could require separating the keys from the data and caching the group key for use on later messages. We informally term the re-use of keys to avoid wasteful repetition of public key operations "recycling."

This protocol is computationally inexpensive, in that it does not require signatures; the worst an attacker could do is provide a faulty key that would soon be discovered. It is also flexible for the group creator, in that the original creator can enumerate any set of friends to include in the group. It is somewhat flexible for others, in that a friend who is a member of two groups ("neighbor" and "football fan") may encrypt a message for the union of these groups ("neighbor OR football fan") by encrypting the message with both group keys separately. However, a friend cannot further restrict access to an intersection ("neighbor AND football fan") without exposing the message to colluding friends that do not match the expression (one a neighbor, the other a football fan). One could encrypt with one group key and then the other, but the colluding members of each set could decrypt the message intended for only the members with both attributes.

Allowing users to encrypt data for groups that they are not members of requires additional infrastructure. Alice can give her friends the ability to encrypt messages for any of her groups defined by an asymmetric keypair by publishing a list of her groups and their public keys. Other users consult this list to send messages to Alice's groups. However, only group members can encrypt to groups defined by a shared symmetric key.

2.2.3 ABE

Alternately, attribute-based encryption (ABE) [9] can be used to implement encryption to groups. To use ABE, each user generates an ABE public key (APK) and an ABE master secret key (AMSK). For each friend, the user can then generate an ABE secret key (ASK) corresponding to the set of attributes that defines the groups that friend should be part of. For instance, if Alice decides that Bob is a "neighbor", "co-worker", and "football fan", then she would generate and distribute to Bob an ABE attribute secret key that includes those three attributes. Bob becomes a member of the groups defined by combinations of those attributes.

In ABE, each encryption must specify an *access structure*: a logical expression over attributes. For instance, Alice can choose to encrypt a message with access structure (‘neighbor’ OR ‘football fan’), where ‘neighbor’ and ‘football fan’ are attributes, rather than groups, and any of her friends who have an attribute secret key with either attribute will be able to decrypt the message. Alice can also encrypt to (‘neighbor’ AND ‘football fan’). In this case, the ABE construction ensures that

only friends with both attributes will be able to decrypt the message. Unlike in the traditional cryptography approach, a single encryption operation constructs the new group and provides the (symmetric) key that protects the rest of the message. Furthermore, any user who knows Alice's ABE public key can encrypt to any access structure (and thus create any group) by knowing the names and definitions of the attributes Alice defined.

ABE provides a natural mapping for the group encryption primitive that we envision for OSNs. This simplicity comes at a performance penalty: ABE operations are about 100-1000 times slower than those of RSA. These ABE operations can be avoided in practice by careful system design. Specifically, ABE defines new groups through attributes and permits sharing efficient, symmetric keys that can be "recycled" to avoid expensive operations. This approach means that ABE's performance penalty need only be paid when it provides its ease-of-use or third-party group-definition advantages, not for each operation.

Consider our example access structure ('neighbor' AND 'football fan'). In creating this group, Alice had to enumerate all her friends and distribute a new group key to matching friends. Now imagine that Bob wants to encrypt data to the same ('neighbor' AND 'football fan') group. With ABE, Bob would encrypt using ('neighbor' AND 'football fan') as the access structure. Under traditional cryptography, if Alice had pre-defined this group (and invited Bob), then Bob could encrypt using the group symmetric key. Otherwise, Bob can encrypt this message *only* if he can enumerate *all* of Alice's friends and know whether they belonged to both groups.

Using ABE allows friend-of-friend interactions without requiring enumerations of friend and attribute lists. A friend may limit who may read a response to a wall post to a more restricted group. For example, if Alice writes "I want to watch Serenity this weekend," as a post to her 'friends', Bob might reply "I have the DVD, let's watch it at my place," to Alice's 'friends' who also have the 'in-the-neighborhood' attribute. Without ABE, Bob would have to rely on Alice to have created this (intersection) group in advance. As long as users share attribute names (and their meanings) with friends, ABE provides an elegant mechanism for users to target information for friends-of-friends. The same functionality can be implemented without ABE, but requires more information exchange (lists of all friends-of-friends and their attributes) and a key distribution mechanism (that maps groups defined by friends to the group key).

2.3 Group Key Management

We describe how Persona users define groups and how users generate and use keys corresponding to groups. Keys guard access to two types of objects in Persona: user data and abstract resources. In Persona, all users store their data encrypted for groups that they define. Any user that can name a piece of data may retrieve it, but they can only read it if they belong to the group for which the data was encrypted. Abstract resources represent non-data objects, for example, a user's storage space or a Facebook Wall. The set of possible operations on an abstract resource is tailored to the resource (for example, it is possible to *write* onto a storage space or *post* to a

user's Wall). Each resource has a *home* which maintains and enforces the resource's Access Control List (ACL). The resource's *owner* may change the resource ACL and allow specific groups different levels of access to the resource. The Persona group management operations described in this section allow users to control access to data and resources. All Persona applications (Section 2.4) are built using these operations.

Each Persona user is identified using a single public key and stores her own (encrypted) data with a *storage service*. We assume for now that users with existing relationships exchange their public keys and storage service locations out of band[2] Storage services support two operations for data storage and retrieval: put and get, which mimic the store and retrieve operations of a hash table. Storage is a resource in Persona, and users may grant other users (or groups) the ability to store (put) onto their storage service using the operations described in this section. Storage services are a specialized case of the broader class of Persona applications and are described in more detail in Section 2.4.1.

We use the notation shown in Table 2.1. In the algorithm listings, $u : \langle \text{protocol step} \rangle$ means user u invokes the specified step.

2.3.1 Operations

Persona operations allow users to manage group membership and mandate access to resources. The operations combine ABE and traditional cryptography,

[2]We describe in Chapter 3 and in Section 4.4.2 how this can be done securely in-band.

Term	Definition
$u.SS$	u's storage service location
$u.K$	Key K created by u
(PK, TSK)	PKC public/secret keypair
$(APK, AMSK)$	ABE public/master secret keypair
ASK	ABE user secret key
$\mathbb{AS}$	Access structure
$\mathsf{TKeyGen}()$	Generate RSA keypair
$\mathsf{TEncrypt}(K, m)$	RSA encrypt m with key K
$\mathsf{TDecrypt}(K, c)$	RSA decrypt ciphertext c
$\mathsf{TSign}(K, m)$	RSA sign m with key K
$\mathsf{ABESetup}$	Generate an attribute public key and master secret key
$\mathsf{ABEKeyGen}(K, attrs)$	Generate an attribute secret key with attributes $attrs$
$\mathsf{ABEEncrypt}(K, m, \mathbb{AS})$	ABE encrypt m with key K and access structure $\mathbb{AS}$
$\mathsf{ABEDecrypt}(SK, PK, c)$	ABE decrypt ciphertext c with secret key SK

Table 2.1: Notation used in this paper.

allowing individuals to be securely added to groups defined using ABE and allowing group members authenticated access to abstract resources.

2.3.1.1 DefineRelationship

Users invoke the DefineRelationship function to add individuals to a group. The user generates an appropriate attribute secret key using the ABEKeyGen function, encrypts this key using the target user's public key, and stores the encrypted key on her storage service. The target user can retrieve this encrypted key using a process described in Section 2.3.3, decrypt it, and use it as necessary.

Algorithm 1 $\mathsf{DefineRelationship}(u1, attrs, u2)$

$u1$: $A \leftarrow \mathsf{ABEKeyGen}(u1.AMSK, attrs)$
$u1$: $C \leftarrow \mathsf{TEncrypt}(u2.PK, A)$
$u1$: $u1.SS.\mathsf{put}(H'(u2.PK), C)$
...
$u2$: $C \leftarrow u1.SS.\mathsf{get}(H'(u2.PK))$

Example Usage: Alice wants to confer the attribute ‘friend’ upon Bob. Alice creates $K = Alice.ASK_{\text{‘friend’}}$, an ABE key associated with the ‘friend’ attribute. Alice computes $C = \mathsf{TEncrypt}(Bob.PK, K)$ after obtaining Bob's public key from out-of-band communication with Bob. Alice stores C on her storage service at the location $H'(Bob.PK)$, where $H'(\cdot)$ is a hash function defined in Section 2.3.3. Bob retrieves C from Alice's storage service and decrypts it, gaining the ability to decrypt content guarded by the attribute ‘friend’. Although any user can retrieve C from its well-known location, only Bob can decrypt it.

2.3.1.2 DefineTransitiveRelationship

The DefineTransitiveRelationship function allows a user Alice to define groups based on a group defined by another user, Bob.

Alice creates a new attribute to describe the new group ‘bob-friend’ and generates an $ASK_{\text{‘bob-friend’}}$ with that attribute. Alice encrypts $ASK_{\text{‘bob-friend’}}$ with the access structure (‘friend’) using Bob's attribute public key and stores the ciphertext on her storage service (Algorithm 2).

Algorithm 2 $\mathsf{DefineTransitiveRelationship}(u1, APK,$ access structure $\mathbb{AS}, attrs)$

$u1$: $A \leftarrow \mathsf{ABEKeyGen}(u1.AMSK, attrs)$
$u1$: $C \leftarrow \mathsf{ABEEncrypt}(u1.APK, A, \mathbb{AS})$
$u1$: $u1.SS.\mathsf{put}(H'(\mathbb{AS}, u1.APK), C)$

Users with the attribute ‘friend’ in Bob's ABE domain may retrieve and decrypt this key and use it to view content encrypted within Alice's ABE domain.

Alice may include a traditional keypair, used for authentication to ACLs, in the ciphertext C. We describe how Bob's friends retrieve these keys in Section 2.3.3.

Example Usage: Alice is advertising a party on an OSN and wants to invite Bob and any of Bob's friends. Alice discovers that Bob uses the attribute 'friend' to define who his friends are. Alice generates the group identity traditional PKC keypair (PK, TSK) for authentication, creates the new attribute 'bob-friend', and generates the attribute secret key $A = \text{Alice}.ASK_{\text{'bob-friend'}}$. Alice calculates

$$C = \mathsf{ABEEncrypt}(Bob.APK, [A, (PK, TSK)], \text{'friend'})$$

and stores it on her storage service at $H'(\text{'friend'}, Bob.APK)$. Alice also performs AssignRightsToGroup to generate group identity keys and instruct the application providing the event advertising service that PK can be used to authenticate RSVPs. Bob sends to each of his friends a link to the application that directs them to Alice's event. Bob's friends cannot initially view the data, so they get C, decrypt it, and view the event. They then get the group identity key, which allows them to authenticate and RSVP to the event.

2.3.1.3 AssignRightsToIdentity

Resource owners use AssignRightsToIdentity to provide other users specific rights to named resources. An example of such a right would be the ability to store data on another user's storage service; we describe other resources and uses in Section 2.4.

To assign rights, the user instructs the resource's home to add a (*public key, set of rights*) pair to the resource's ACL. If the public key was already in the ACL, then the rights are changed to those specified in the new rights set (Algorithm 3).

Algorithm 3 AssignRightsToIdentity($u1, rights$, PK, resource r, owner o)

$u1$: o.chACL($r, PK, rights$)

User $u2$ who possesses *TSK* may exercise the named rights on the resource by authenticating to the resource's home node using *TSK*.

Example Usage: Alice wants to give Bob the ability to put data on her storage service. Alice instructs her storage service to create a new ACL rule based on *Bob.PK* that allows write access. Bob later calls the put function on the location L with the world readable data m. Alice's storage service issues a nonce n, and Bob replies with TSign($Bob.TSK, [n, \text{“}write(L, m)\text{”}]$). Alice's storage service verifies the signature against *Bob.PK*, authenticating Bob's write according to Alice's access policy.

2.3.1.4 AssignRightsToGroup

The AssignRightsToGroup function allows a user Alice to provide resource access to a group G rather than to an individual. The group is specified using attributes defined in Alice's ABE domain.

First, Alice creates a new (PK,*TSK*) pair specifically for G. Alice ABE-encrypts this keypair with an access structure that identifies members of G. Alice stores the resulting ciphertext on her storage service. This pair of PKC keys becomes

the group identity and Alice can assign rights according to AssignRightsToIdentity. The pseudocode is presented in Algorithm 4.

Algorithm 4 AssignRightsToGroup($u1, rights$, access structure $\mathbb{AS}$, resource r, owner o)

$u1$: $(PK, TSK) \leftarrow$ TKeyGen()
$u1$: $C \leftarrow$ ABEEncrypt($u1.APK, (PK, TSK), \mathbb{AS}$)
$u1$: $u1.SS$.put($H'(\mathbb{AS}, APK), C$)
$u1$: AssignRightsToIdentity($u1, rights, PK, r, o$)

Example Usage: Alice wants to give her friends and her family the ability to put data on her storage service. Alice defines the group G as the users who have 'friend' or 'family' in their ASK in Alice's ABE domain. Alice creates $K = (PK_G, TSK_G)$, and stores

$$C = \text{ABEEncrypt}(Alice.APK, K, (\text{'friend' or 'family'}))$$

on her storage service. Anyone who possesses either of these attribute keys can retrieve C, decrypt it with their ASK, and use TSK_G to authenticate to store data on the storage service as described in AssignRightsToIdentity.

2.3.2 Revocation of Group Membership

Removing a group member requires re-keying: all remaining group members must be given a new key. Data encrypted with the old key remains visible to the revoked member. The nominal overhead is linear in the number of group members but it may be possible to reduce it [92].

An ABE message can be encrypted with an access structure that specifies an inequality ("keyYear $<$ 2009"), and the message can be decrypted only if a user

possesses a key that satisfies the access structure. This facility can be used to provide keys to new group members such that they cannot decrypt old messages sent to the group.

2.3.3 Publishing and Retrieving Data

Private user data in Persona is always encrypted with a symmetric key[3]. The symmetric key is encrypted with an ABE key corresponding to the group that is allowed to read this data. The group is specified by an access structure as described in Section 2.2.3. This two phase encryption allows data to be encrypted to groups; reuse of the symmetric key allows Persona to minimize expensive ABE operations.

Users put (encrypted) data onto their storage service and use applications to publish references to their data. Data references have the following format:

$$\langle \text{tag, storage service, key-tag, key-store} \rangle$$

The tag and storage service specify how to retrieve the encrypted data item, and the key-tag and key-store specify how to obtain a decryption key.

Users read data by retrieving both the item and the key. Suppose item i is encrypted with symmetric key s. If user u_1 wants to read i and u_1's local cache or own storage service does not contain s, u_1 can retrieve the ABE-encrypted s using the key-tag and key-store information in the reference. s is encrypted under the access structure $\mathbb{AS}$ in the ABE domain defined by APK (u_1 can infer both from the encrypted key). u_1 tries to decrypt s using her ABE secret key, and if successful,

[3]Users may store public data in plain-text to reduce overhead.

decrypts i using s. u_1 stores s, encrypted with her own public key, on her own storage service for future use. The encrypted key is stored at $H(\mathbb{AS}, APK)$, where $H(\cdot)$ is a hash function. If s is instead associated with traditional public key PK, u_1 stores the encrypted s at $H(PK)$.

Suppose user u_2 wants to encrypt a message for a set of users specified by access structure $\mathbb{AS}$ in the ABE domain with public key APK. The domain may belong to u_2 or to some other user; u_2 only needs to know the public parameters for this domain in order to encrypt.

u_2 looks for a symmetric key for this group by invoking $u_2.SS.\mathsf{get}(H(\mathbb{AS}, APK))$. Such a key would exist if u_2 had previously encrypted or decrypted messages for this group. If the retrieval succeeds and the encrypted symmetric key is found, u_2 decrypts it using his own public key and obtains the symmetric key s.

If the retrieve fails, u_2 constructs a new symmetric key s, encrypts it with his own PKC public key and stores it in $u_2.SS$ under the tag $H(\mathbb{AS}, APK)$. u_2 further encrypts s using $\mathsf{ABEEncrypt}$ with access structure $\mathbb{AS}$ and APK and stores this ABE-encrypted symmetric key on $u_2.SS$ with the tag $H'(\mathbb{AS}, APK)$. H' is a hash function different from H. By construction, the ABE-encrypted key can be decrypted exactly by those users who belong to the group to which the message is encrypted. This group may not include u_2. If u_2 wishes to encrypt s with traditional PKC instead of ABE, u_2 encrypts with public key PK and stores the encrypted key at $H'(PK)$.

Finally, u_2 encrypts the message using s and stores it using tag M. u_2 can

then publish a reference to this item of the form:

$$\langle M, u_2.SS, H'(\mathbb{AS}, APK), u_2.SS \rangle$$

Other users resolve the reference by invoking $u_2.SS.\mathsf{get}(M)$ which will retrieve the original message encrypted with s.

In this example, u_2 obtained the decryption key from his own storage service (or created a new key and put it on his own storage service). In general, however, u_2 may already know a different key for this group (for example, one that was used by a different user to encrypt to the same group) that is stored on some other storage service. Instead of creating his own key, u_2 may choose to refer to this pre-existing key instead.

2.4 Applications

Persona users interact using applications. Even core functions of current OSNs, including the Facebook Wall or Profile, exist in Persona as applications. In this section, we describe how applications use the group key and resource management operations of Section 2.3.

Persona applications export a set of functions (an API) and a set of resources over which those functions operate. When there are resources, such as file stores or documents, two functions are expected in the API. First, register allocates a resource for a principal (to create a Wall, for example). Registration with an application returns a reference to the newly-allocated resource to the client. Second, chACL allows the owning principal to define access restrictions via ACLs: for a given resource and

a given principal, permit an operation. Applications will support further operations, as we describe below, starting with the basic storage service.

2.4.1 Storage Service

Storage is a basic Persona application that enables users to store personal data, make it available to others who request it, and sublet access to storage for applications to use for per-user metadata. A user trusts a storage service to reliably store data, provide it upon request, and protect it from overwrite or deletion by unauthorized users. A user does not trust a storage service to keep data confidential, relying instead on encryption to guard private information.

The storage service exports both get and put functions. The storage application returns data whenever the get is invoked with a valid tag. The invoking principal is not authenticated or validated, since the expectation is that data is protected via encryption.

The put function requires the invoking principal n to authenticate to the storage application. When n wants to put data, she presents her public key K and the store identifier s to the storage application. The storage application ensures that (K, put) exists in the resource ACL corresponding to s, and authenticates n using a challenge-response protocol. n may write into s if the authentication succeeds.

Applications must store the metadata they have constructed. They can provide their own storage or use a storage service. If the application provides its own storage resource, the application returns a handle to the resource when a user reg-

isters with the application. The user can then call AssignRightsToIdentity to give other users access to the application's storage resource.

The user can instead provide the storage resource to the application and invoke:

$$\mathsf{AssignRightsToIdentity}(user, \texttt{write}, App.PK, c, user.SS)$$

where c is a storage resource on $user.SS$, to allow the application to write onto the user's storage server. The user now registers with the application, passing it the storage resource c in which to store the metadata:

$$R \leftarrow App.register(user.PK, c)$$

In turn, the application returns a reference (R) to the resource corresponding to the application instance.

To prevent an attack in which another user u_2 pretends to own c, the registering user must prove that he owns c. He does this by writing a nonce provided by the application into c. The application ensures the nonce is present before writing.

2.4.2 Collaborative Data

The predominant method of sharing data in OSNs is via collaborative multi-reader/writer applications. For instance, the quintessential Facebook application, the Wall, is a per-user forum that features posts and comments from the user and her friends, the Facebook Photos application stores comments and tags for each picture and displays them to friends, the MySpace comments section allows friends to write to a user's page and read others' comments, and each photograph posted to Flickr

has a page where members of the Flickr community can comment on photographs. Instead of re-implementing each OSN application in Persona, we present a generic multi-reader multi-writer application named Doc. Doc can be used as a template for implementing a variety of OSN applications, as we describe in Sections 2.4.2.1–2.4.2.4.

Doc is organized around a document shared between collaborating users. Users register with the Doc application and create a new *Page*. The application associates a resource with this Page, and allows the user to provide read or write access to other users (or groups). The Page metadata contains references to encrypted data; the application is responsible for formatting this data for display. Users who are allowed to write to the Page contact the application with data references, and Doc updates the Page appropriately. The Page can be stored by the application or on a storage server specified by the original user (in which case the user has to provide the Doc with write access to the Page stored on the storage server). We describe these steps next.

Reading the Page. To allow Bob to read content in her Page, Alice must give Bob appropriate keys and a reference to her Doc. In particular, Alice must provide an attribute secret key *ASK* that will allow him to decrypt (some subset of) the data in the Page. Alice decides which attributes Bob should get and calls

$$\mathsf{DefineRelationship}(\text{Alice}, attrs, \text{Bob})$$

to issue an *ASK* to Bob. Obviously, Alice may already have given Bob these attributes, in which case this step can be skipped. In either case, she provides him

with a reference to her Page.

Bob can now retrieve the Page metadata, resolve data references, and decrypt (potentially only a subset of) the Page data.

Writing to the Page. Alice may want to provide Bob with the ability to write to her Page, where writing is a function exported by the Doc application. She does so by adding Bob's public key to the Page's resource ACL by invoking:

$$\mathsf{AssignRightsToIdentity}(\text{Alice}, \texttt{write}, \mathit{Bob.PK}, D, \mathit{Doc})$$

Bob may now write onto the Page. Bob stores (appropriately encrypted) data onto a storage-server and notifies the Doc of a write onto Alice's Page. The Doc application must authenticate Bob and ensure that his public key is in Alice's Page's ACL with the proper right. If the authentication succeeds and Alice has provided Bob the write right, then the Doc application updates the Page metadata (either stored at the application or on a storage server specified by Alice) with the data reference provided by Bob. The interpretation of the Page metadata is application-specific.

Alice may authorize multiple users to write to the same Page. Conflicting updates or concurrent writes are handled by the Doc application, possibly by storing the Page as an append-only log. Users need not encrypt using a single access structure, and may choose any access structure they desire. They may even write onto a Page using an access structure that cannot be decrypted by some of the Page's readers.

In summary, Doc is a general multi-reader/writer template for storing and

formatting metadata with references to encrypted content. Doc can easily be tailored to implement many useful OSN applications, as we demonstrate next.

2.4.2.1 Wall using Doc

The Facebook Wall is a multi-user collaborative application that allows a user's friends to read messages, post messages, and comment on posts onto a shared document, called the user's Wall. Doc can be used to (almost trivially) implement the Wall application. Unlike the Facebook Wall, the Persona Wall is distributed: it allows users to choose where the Wall metadata is stored. All posts and comments are stored on storage servers owned by the poster/commenter. The Wall document itself contains rendering information and references to writes onto the wall. These references must be resolved (i.e., the data fetched from appropriate storage servers) and decrypted before rendering the Wall. End-user applications may intelligently cache data and keys to reduce rendering latency.

2.4.2.2 Chat and Status Updates over Doc

A chat application can use Doc as the template. A chat session is a shared document to which the chat host invites other users (and provides them write access to the chat Doc). The chat application has to implement auxiliary UI functions (such as an invite notification, and polling for new messages), but the basic structure follows that of a simple Doc onto which users may append messages.

Doc can also be used to implement user-specific status updates. The user creates a status Doc and provides read-only access to other users (or groups) who can periodically read the Doc to receive updates. The reference to the status update Doc may be obfuscated such that unauthorized users are not able to detect changes in status (even if they are not able to decrypt the status message).

2.4.2.3 News Feed using Doc

The news feed in Facebook collects "stories" from other applications to provide a temporal view of Facebook activity. In Persona, the user provides the news feed with a list of applications that she wants to appear in her feed, and an *APK* and $\mathbb{AS}$ (or perhaps several access structures along with a policy dictating when to use each access structure) with which to encrypt the feed. Only the user may change the list of monitored applications. The news feed application retrieves the metadata from the selected applications and parses it to create a history of changes to the user's applications' metadata. The application writes this history as a user would write a Page; only the news feed may write to this metadata. Viewing the feed consists of viewing the Page. The contents of the Page are visible to anyone that can satisfy $\mathbb{AS}$.

2.4.2.4 Other Applications

Other popular Facebook applications such as Profiles, Photos, Groups, and Events can be implemented using Doc as well. These applications can be imple-

mented by altering the interpretation and presentation of metadata and tailoring the API to the relevant task. Though Doc is sufficient for many Facebook applications, we consider examples of existing applications that require additional features in the following sections.

2.4.3 Selective Revelation

The user may want to share some personal data with an application. One such example is an application that allows users to search for others. Alice can choose exactly the information by which other users can find her by only sharing that data with a Search application. Another example is the *Where I've Been* Facebook application [117]. Users enter a list of countries or cities that they have lived in, visited, or want to visit, and the application shows a map with these locations highlighted. Users can also compare maps with another user to see which locations they have in common.

In order to permit applications that post-process personal data, we allow them to decrypt certain data by giving them an *ASK*. Alice encrypts a list of cities she has visited with the access structure (`'classmates' or 'where-ive-been'`). She generates an *ASK* and encrypts it with the Where I've Been application's *PK*:

DefineRelationship(Alice, 'where-ive-been', Where I've Been)

When she registers to use the application, she gives it a reference to the encrypted key. The application retrieves the key and can now decrypt and parse Alice's list of cities to produce the highlighted map. This general approach of selectively revealing

user data to applications has been discussed earlier in [47], and, since the publication of this work, has been integrated into modern OSNs and mobile devices in the form of permission requests during application installation.

Application functionality that can be implemented without revealing personal information is surprisingly broad; however, in some cases, the application must compute transforms over the user's data. This is the case for the Where I've Been application, especially when it has to compare the locations of multiple users. We return to the general problem of structuring private applications and the tussle between application functionality and user privacy in Section 2.7, and describe a general framework for untrusted third-party assisted rendezvous in Chapter 4.

2.4.4 Applications that use the social graph

The graph of social connections between Persona users is not public. It is realized only in the collections of public keys of friends a user stores, and given meaning only through the assignment of attributes using DefineRelationship. This obscurity of friend links frustrates applications such as those that analyze the graph of connections to help connect with more friends (People You May Know) or to visualize interconnections between friends (the Friend Wheel).

To enable these applications, users have two options. A user may publish social links to each application using selective revelation or by directly uploading a set of relationships. Alternatively, a single, somewhat trusted social link application might provide access to other applications.

Published edges in the social graph are protected just as other data in Persona: encrypted to be hidden from arbitrary users and applications, but exported to chosen users and useful applications that may access only what they require.

2.4.5 Inherently private applications

Persona allows for potential applications which are not realistic on OSNs without privacy. For instance, a user might want to have a Medical Record application where she stores her medical data. She might not want her employer or her friends to see her data, but she would want to share it with her doctor. She may even have many doctors, and it may be helpful for them to collaborate in a central location. There is no technical difference between this application and Doc. However, these applications are uniquely available on Persona because they operate on sensitive private data.

2.5 Implementation

Our Persona implementation consists of two Persona applications (a storage service and a customizable Doc application) and a browser extension for viewing encrypted pages and managing keys.

2.5.1 Storage Service Application

Our Persona storage service application is an XML RPC server using PHP and Apache with a MySQL database backend. The service implements the storage

API described in Section 2.4.1.

2.5.2 Doc Application

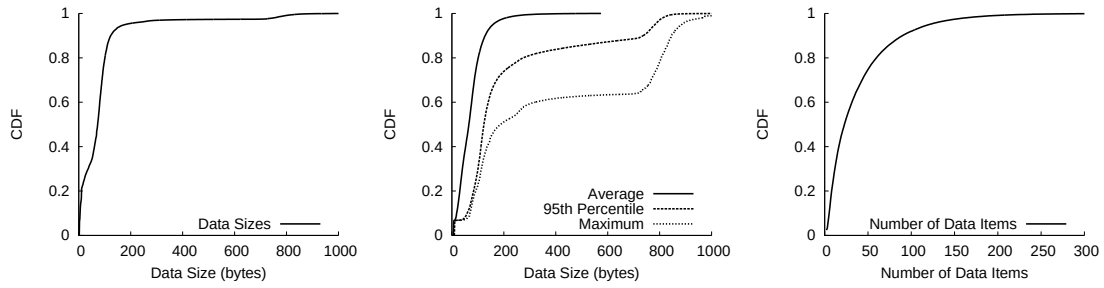
We have implemented a Doc application (Section 2.4.2) in PHP with a MySQL backend for storing metadata. Using the Doc as the base, we implemented Profile and Wall applications.

Our Profile application presents an interface for the user to put data onto her profile and read others' profiles. The profile metadata (stored by the Profile application in a MySQL database) consists of references to encrypted profile data items. The Profile application allows only the registered user to write onto the DocPage.

Our Wall application is identical in structure to the Profile, but allows other users to write onto the Doc as well. The Wall application allows users to post new items and reply to existing items. The Wall application constructs the Wall Doc metadata file threading posts and replies. As with all applications, the posts and references themselves are stored on other storage services, and the Wall application operates using item references only.

2.5.3 Browser Extension

Users interact with Persona using a Firefox extension. The extension uses the XPCOM framework in the Mozilla Build Environment to access the OpenSSL and cpabe [99] libraries for cryptographic operations. The extension allows users to



(a) (b) (c)

Figure 2.1: (a) CDF of the size of Facebook profile data items. (b) CDF of the maximum, 95th percentile, and average of the size of data items per Facebook profile. (c) CDF of the number of data items on Facebook profiles.

register with applications, encrypt data to groups, resolve data references, decrypt data using appropriate keys, and facilitate out-of-band public-key exchange.

The browser extension is a trusted component in Persona; it is, in fact, the only one. The extension implements a secure keystore, to which users upload their private and public keys. The extension is given a list of public keys corresponding to the user's contacts. These keys are also stored (encrypted with the user's public key) on a storage service. When a user uses a new browser, the extension is initialized with the user's private key and a reference to the user's permanent keystore. The extension then downloads all of the other keys from the storage service.

When an encrypted Persona page is loaded, the extension processes the elements on the page and replaces them inline if necessary. There are two main types of replacement: resolution of data references and replacement of special tags.

Data reference resolution. The extension parses item references, fetches the items from storage services, decrypts the items, and verifies any signatures on

those items. In our implementation, all data is signed by the creator and verified if the signer's key is known. Data resolution is recursive: encrypted data may contain references to more encrypted data.

Our extension uses an XML-RPC javascript library capable of sending asynchronous RPCs. During page processing, all data items are fetched asynchronously using `XMLHttpRequest`. If the items are encrypted with an unknown key, the keys are also fetched asynchronously. Once all keys and data items have been fetched, the extension sequentially decrypts (and verifies) each item, and replaces the references with the decrypted text. We are currently extending our implementation to decrypt items as they arrive rather than waiting for all fetches to complete.

Replacement of special tags. Persona users may not want to share their list of contacts (to be precise, their public keys) with applications. Instead, this list is kept encrypted with the user's public key on a storage service, which the extension downloads upon initialization. The extension recognizes a "friend-form" tag sent by an application, and replaces this with a drop-down box containing a list of the user's contacts. This facility is used in our Profile application to allow a user to view their contacts' profiles.

The extension allows users to encrypt data to groups. It replaces embedded forms with a text box into which the user can enter private data. When the submit button is pressed, the extension prompts the user for a policy under which to encrypt the data, performs the encryption (constructing and publishing symmetric keys as necessary), puts the encrypted data on the user's storage service, and replaces the form data with a reference to the encrypted data item.

Caching. To reduce latency, the extension caches various keys and contact information. This includes keys the user has created: an RSA public key (137 bytes for 1024-bit moduli), RSA private key (680 bytes), *APK* (888 bytes), and *AMSK* (156 bytes). For each friend, the extension caches their storage service information, RSA public key, and *APK*. The extension also stores the *ASK* (the size varies: 407 bytes for one attribute and 266 bytes for each additional attribute) created for that friend along with the attributes associated with the *ASK*. For each policy that the user is a part of, whether it is created by the user or a friend, the extension caches the RSA keypair and the symmetric key.

This caching and recycling of symmetric keys allows the extension to pay the cost of an ABE decryption only when it encounters an item encrypted using a new key reference. This will occur when the encryption uses a new policy (corresponding to a new group) or an existing policy to which a user has encrypted with a new symmetric key. The latter might occur if the encrypting user is not part of the group and is unable to decipher existing symmetric keys for that policy. The common operation of the extension does not require expensive ABE operations.

2.5.4 Integrating Persona with Facebook

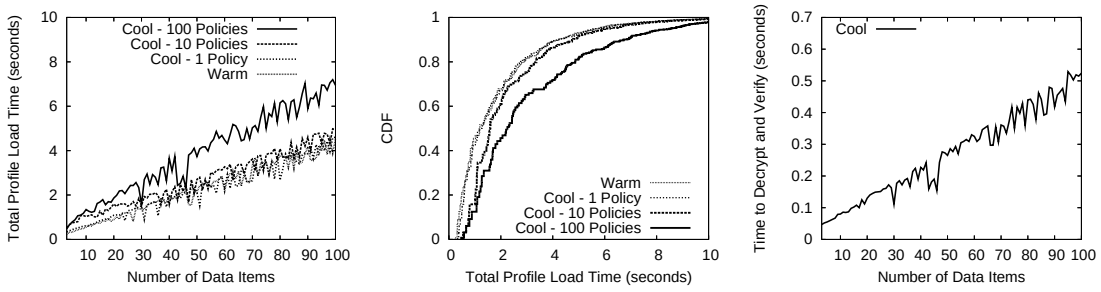
Current deployments of OSNs underline their undeniable popularity. It is not realistic to assume that Persona (or some other privacy-enabled network) will replace existing OSNs. Instead, we expect users to migrate personal information onto private networks, while continuing to use existing OSNs for public data.

We have designed Persona to inter-operate with existing OSNs, and our prototype integrates with Facebook. Persona applications are accessible as Facebook applications and can interact with Facebook's API, providing privacy-enabled applications through the familiar Facebook interface. Conversely, existing Facebook applications can be made Persona-aware on a per-application basis. Users protect their private data by storing it on Persona storage services rather than on Facebook; only fellow Persona users will be able to access the data, and only if they are given the necessary keys and access rights.

Using Persona applications within Facebook. Users log-in to Persona by authenticating to the browser extension (which then decrypts and encrypts data transparently), and then log-in to Facebook as normal. A Facebook-aware Persona application is akin to any third-party Facebook application, and can be selected for use as any other Facebook application. Unlike other applications, Persona applications use markup that is interpreted by the Persona browser extension, and are aware of data references.

Traditional Facebook applications may use the Facebook API to communicate to users by sending notifications, displaying items on the Facebook wall, and sending application invitations. The same facilities are available to Persona applications. We have implemented an abstract OSN interface that Persona applications use to access OSN APIs. While our design is general, our current implementation has only been tested with Facebook. Our Doc-based applications are accessible via Facebook as Facebook applications.

Using Facebook applications on Persona. Once users begin to use Per-



(a) (b) (c)

Figure 2.2: (a) Total time needed, in seconds, to present Facebook profiles composed of encrypted data items. (b) CDF of total time to load Facebook profiles. (c) Total time needed, in seconds, to decrypt encrypted data items in Facebook profiles in the *cool* data set with 100 groups. Note the difference in scale from (a).

sona, existing Facebook applications may want to provide Persona users with the ability to store private data. Minimally, each application has to be ported to operate using Persona data references, though some applications that transform user data may require a complete rewrite. We discuss application porting in Section 2.7.

2.6 Evaluation

In this section, we quantify the processing and storage requirements of Persona and measure the time to render Persona-encrypted web pages.

The key parameters of our evaluation are the sizes and number of distinct data elements that might be stored on a single Persona page. Each distinct element represents a request to a storage server and may, if the policy and associated key are unknown, also imply a request for a group key and its decryption with ABE. This process represents the performance cost of Persona. We estimate these param-

eters using Facebook as a model, combining real user profiles from Facebook with observations of application-provided limits on the number of items per page.

User profiles can contain hundreds of data items. We use profile data in our evaluation because it exposes the worst case performance of Persona, where users must fetch and decrypt many individually encrypted data items. Our data is from a crawl of Facebook profiles gathered in January, 2009. The crawl contains the HTML of the profile pages of 90,269 users in the New Orleans network; of those pages, 65,324 pages contain visible profiles, and 39 pages had miscellaneous errors that left them unusable.

We parse these Facebook profiles into data items that could be individually encrypted. First, we parse the document based on fields such as Name, Birthday, Activities, Interests, etc. We then decompose fields which contain multiple items separated by commas, bullet points, or line breaks. Under this decomposition, users would be able to, for example, individually encrypt every TV show, book, and movie that they enjoy, if they chose to do so.

Figure 2.1 (a) shows a CDF of the sizes of all data items and Figure 2.1 (b) shows a CDF of the maximum, 95th percentile, and average data item sizes on a per-profile basis. These plots show that most of the data items are small, but many pages also have a few large items. We also present a CDF of the number of data items per profile in Figure 2.1 (c). These figures provide a backdrop for the performance of Persona: our results show that the number of data items on a page determines the page load time.

2.6.1 Desktop Performance

We evaluate our Persona implementation on a desktop computer using a 2.00 GHz processor and 2 GB of RAM. The desktop, storage service, and application server are connected through a router which introduces an artificial delay, chosen uniformly between 65ms and 85ms, on each packet. These values reflect high latencies observed by King [32] and represent a case where the storage service is far away from the user.

We use two experiment scenarios. The first, termed *cool*, represents Persona in its initial state, when group symmetric keys must be retrieved from a storage service and decrypted. The second, termed *warm*, represents Persona usage in the steady state, when all symmetric keys associated with groups have been cached. We repeat the *cool* experiment scenario three times, varying the number of user-defined groups between 1, 10, and 100. We run only one *warm* experiment scenario since no key fetches and no ABE decryptions are needed. In each data set, we randomly assign each data item to one of the user-defined groups.

For each Facebook profile, we first encrypt and store each of the data items in Persona. We then retrieve a page that contains references to all of these data items. In the *cool* data set, we asynchronously fetch the keys needed to decrypt all of the items in the page. In both *cool* and *warm*, we also asynchronously fetch the encrypted data items themselves. Once all keys and data items have been fetched, we decrypt the data items on the page, verify their signatures, and re-render the page. For efficiency, rather than evaluating every profile, we evaluate a profile page

drawn randomly from the set of all pages that have x items, for all values x for which there is a profile with x items.

Page load time. Page load times increase linearly with the number of elements. Figure 2.2 (a) shows how long it takes to download, decrypt, and display the profile page for each of our experiments, as a function of the number of data items on the page. We extrapolate the distribution of page load times per Facebook profile in Figure 2.2 (b). The median page load time is 2.3 seconds and the maximum is 13.7 seconds. Most pages consist of a few, small entries, so most are loaded quickly. The *cool* data sets are comparable to the *warm* data set, indicating that retrieving keys is not too expensive. These times may also represent a worst case; if users aggregate their data more coarsely there will be fewer data items, requiring fewer fetches and thus fewer round-trip times. Another possible improvement would be to cache commonly retrieved data items, but we have not performed this optimization.

Encrypted data size. We show how much larger the encrypted data is for individual data items in Figure 2.3a and for entire profile pages in Figure 2.3b. There is a substantial increase in the size of the stored data, and this will affect both the storage capacity of the storage services and the network resources required to transfer data. The storage services are inherently distributed, so they should be able to scale to support the needs of the system.

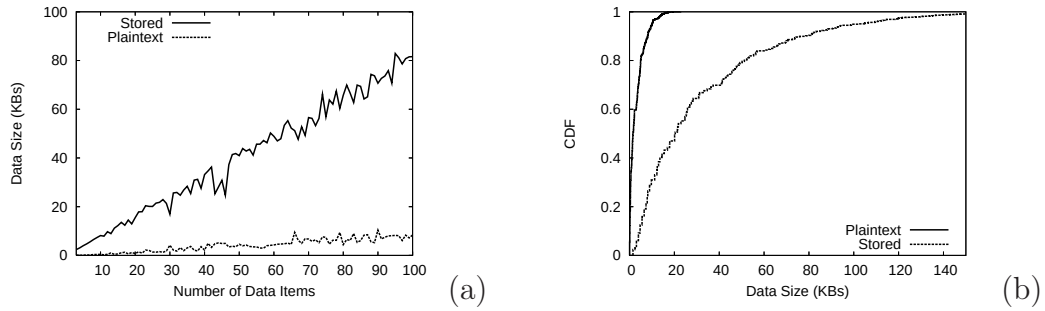


Figure 2.3: (a) Total size of plaintext and stored (ciphertext and signature) data for Facebook profile pages by number of data items on the page. (b) CDF of total size of plaintext and stored (ciphertext and signature) data for Facebook profile pages.

2.6.2 Mobile Device ABE Performance

Mobile devices are increasingly used for limited access to OSNs. MySpace, Facebook, and LinkedIn [112] all have iPhone applications, and there are many twitter and instant messaging clients. Persona, to provide a substitute, must also be realizable on mobile devices. Enabling mobile devices with Persona-based security would enable users to exchange their current locations with friends but not third parties, enabling functionality similar to that of Loopt [113] without trusting the service provider.

The requirements for encryption performance of mobile OSN clients are a bit different from their desktop counterparts. Because of their smaller screens and often slower network connections, the requirements for decryption are less demanding: when only a few messages may be retrieved or displayed at a time, decrypting only a few items is necessary. Conversely, mobile devices tend to have limited computation power and limited battery life, so the operations themselves should be reasonably interactive.

We cross-compiled the cpabe [99] libraries and their dependencies (pbc [53], gmp, glib, openssl, gettext, libiconv, and libintl) for the iPhone SDK 2.2.1 [101].[4] Some of these libraries (e.g., libcrypto from OpenSSL) are present on the device but not included in the official SDK. Cryptographic operations supported on the device may be implemented using hardware acceleration when applications are written using Apple's defined APIs; writing directly to the OpenSSL library forgoes these potential advantages. In other words, our benchmark is sufficient to show that ABE is practical on a widespread mobile device, but not intended to compare ABE or AES performance from one device to another.

On a first-generation iPhone (620MHz ARM), decryption of ABE encrypted text fragments smaller than 1KB takes approximately 0.254 seconds. This value is the average time to decrypt 40 randomly-generated messages of 40 different sizes drawn uniformly at random from 0 to 4095 bytes of 5 different access structures having one to five attributes. Message size and access structure have little effect: the message itself is encrypted using AES-128, and the access structure appears to have a greater effect on the time to encrypt than to decrypt. Encryption times average 0.926 seconds with one attribute (an average of 25 messages of 25 sizes; some of this time is likely consumed by AES-128 key generation) and 0.43 seconds for each additional attribute.

We believe that the 0.254 second object decryption time compares favorably to the typical RTT of cellular data systems (Lee [50] reported a 417ms average RTT

[4]Patches to enable cross-compilation of these libraries using Apple's gcc compiler are available at http://www.cs.umd.edu/projects/persona

for 1x EV-DO) and does not preclude a mobile Persona.

2.7 Discussion

Our Persona prototype and evaluation demonstrates new functionality and reasonable performance. In this section, we discuss unexplored questions a large-scale deployment will have to confront.

Factoring applications. Persona was motivated by the observation that current OSN applications have complete access to user data. Current Persona applications, on the other hand, have no access to user data and must operate entirely using data references. Applications that act on user data must be given selective access as described in Section 2.4.3. This approach is similar to how others [47] have discussed statically classifying user data in OSNs for application access.

An alternate design is to refactor applications into one piece administered by the application provider (as now), and another piece capable of transforming user data that would be executed on a trusted host (likely, within the user's browser). Existing taint-tracking techniques [79, 95] can be used to guarantee that user-data remains safe. This option relieves the user from thinking about what data should be released to which applications; however, application design and implementation must undergo a substantial change.

Factored data. Persona decouples application metadata from encrypted content. This may lead to cases when one is available but not the other. Ideally, data and metadata would share availability, but combining both might lead to unacceptable

performance or violate storage policy (about where data might be stored). A scalable policy-compliant design for a fate-sharing [17] dissemination infrastructure is an open problem.

Deployment incentives. OSNs are popular, in part, because they are free. Persona's design requires users to contract with applications, and some applications, such as the storage service, may have little incentive to provide free service. Users may have to pay for this storage or agree to use some other service or applications in exchange for free storage. Other applications—for instance, versions of Doc—may augment the metadata with advertisements, which may provide a sustaining deployment model. As privacy-enhanced OSNs become popular, current OSN providers may choose to incorporate privacy features, in effect supporting the Persona + Facebook model we have implemented.

2.8 Conclusion: Persona in Perspective

Privacy controls provided by existing OSNs are not sufficient since they rely on trusting the OSNs with data from which they can profit. We have shown how ABE and traditional public key cryptography can be combined to provide the flexible, user-defined access control needed in OSNs. We have described group-based access policies and the mechanisms needed to provide decryption and authentication by both groups and individuals. We have demonstrated the versatility of these operations in an OSN design called Persona, which provides privacy to users and the facility for creating applications like those that exist in current OSNs.

Persona was among the earliest work to solve the problem of privacy in OSNs, and, as OSNs are a part of a rapidly developing industry, much has changed in the three years since the original publication of the work. Privacy controls in popular, centralized OSNs have improved to satisfy user complaints. In particular, the launch of Google+ saw the introduction of "circles" for controlling the privacy of posted information; these circles were well-received and vindicate the design decision of attribute-based grouping in Persona. However, despite their improved privacy controls, today's popular OSNs still rely on a centralized service that the user must trust in order to participate. Persona and other decentralized OSNs provide a level of privacy and security that cannot be provided by a centralized service, by providing users the ability to choose with whom they will store their data and (through local cryptography) how that data will be stored.

Chapter 3

Bond Breaker

3.1 Introduction

OSNs have persuaded millions of users to give their offline identities an online presence. While these OSN identities are convenient for online communication, they risk impersonation [98] and may provide personal information that threatens the security of other systems [71, 114]. Users, aware that their personal information is valuable, may choose only to allow their friends to see their information. However, even correct privacy settings can be foiled if someone has infiltrated their circle of friends. Users cannot trust that the person behind an online account is actually their offline friend, even if that account has the correct picture and profile information [11]. Solving the problem of OSN impersonation is necessary to establish a secure, privacy-enabled OSN such as Persona (Chapter 2).

Many modern applications allow users to authenticate using an OSN account. Authenticatr [72] shows that these identities can be a valuable tool in system design. Unfortunately, an OSN provider is not equipped to verify user identities since the provider knows almost nothing about its users other than what they themselves supply, and that supplied data can be easily forged. Though OSN users are also not able to identify arbitrary OSN users, they are actually well-equipped to detect when an attacker is impersonating one of their friends.

Offline users have ways of identifying a friend—such as recognizing her appearance or voice—that are either difficult or impossible in online communication. Instead they can use *exclusive shared knowledge* for identification: they can identify a friend (either online or offline) by asking questions that only she can answer.

Once the user identifies his friend, he can ask her to provide or verify a public encryption key associated with her identity. By repeating this process with all of his friends, the user bootstraps a public key infrastructure (PKI) that he can use on the OSN, a PKI that is important for emerging OSN applications that require security or privacy.

We[1] face several challenges by verifying OSN identities with shared knowledge. We must guarantee that shared knowledge remains secret or we open ourselves up to impersonation attacks. Users may not share exclusive knowledge with all of their friends, so the PKI we create may be limited in scope. Lastly, an impostor may be able to guess the knowledge shared by a pair of users, so we must limit and, if possible, detect such attacks.

Our contributions are the following. We show that existing protocols can be used in an OSN to exchange keys without revealing shared knowledge. We perform a user study that shows that strangers have less than a 2% chance of guessing the answers to shared knowledge questions; this compares favorably to web-based security questions—another identification scheme based on personal information—which can be guessed 17% of the time by strangers [77]. We show that even when users only exchange keys with a few friends, we can discover the keys of many friends

[1]This work [7] involved the collaboration of Bobby Bhattacharjee and Neil Spring.

and friends-of-friends with a web of trust [81]. Finally, we show that the same web of trust detects 80% of all successful impersonation attacks.

We organize this chapter as follows. In Section 3.2 we describe how to use exclusive shared knowledge to distribute public keys and show that we can avoid impersonation attacks with existing protocols. We describe our user study in Section 3.3 and show that shared knowledge exists and can be used to identify friends. We describe related work in Section 5 and conclude in Section 3.4.

3.2 Exclusive Shared Knowledge

The strength of exclusive shared knowledge lies in its secrecy, so we must handle it delicately to prevent attacks. We seek a key exchange protocol in which one user can use shared knowledge to verify another user's offline identity, without either user revealing that knowledge in the process.

3.2.1 Design

One user, the *asker*, wishes to verify the identity of her friend, the *askee*. The users are communicating over an insecure channel and we assume their messages can be intercepted by a man-in-the-middle attacker, the *meddler*. Key exchange is asymmetric: in one instance of the protocol, the asker identifies the askee only. Symmetry is not required in OSNs that have directed friend relationships, such as Twitter. For symmetric OSNs like Facebook, we realize symmetry by repeating the asymmetric protocol with the asker and askee roles reversed.

We will apply exclusive shared knowledge in our protocol as follows. The asker formulates a question Q with answer A that relies on the exclusive knowledge shared between the asker and askee. At the end of the protocol, the asker will receive a public key PK with the guarantee that the person who sent the key used the answer A in the protocol, even though A is never communicated in any way.

3.2.2 Attacks

We first consider the askee impersonation attack. The meddler, though he does not know A, may make a guess G that could be equal to A, especially if the set of possible answers to Q is small. The meddler will attempt to use G to offer the fake key FK instead. If $G = A$, the asker will receive FK and be convinced that it belongs to the askee, meaning the impersonation is successful. However, if $G \neq A$, the asker will be unable to verify the askee's identity and may grow suspicious of an impersonation attempt.

A meddler who can prevent messages from being delivered could also prevent successful verification of the askee. The general problem of denial of service attacks is outside of the scope of this work; we expect that existing techniques can be applied straightforwardly to this setting.

Alternatively, the meddler could attempt to impersonate an asker rather than the askee. The meddler chooses a question Q' and asks it of the askee. The askee does not reveal A' in the protocol, so the meddler can only learn A' if her guess G' is correct. A' is only useful information in a subsequent askee impersonation attack,

and even then only if the asker chooses to use Q' as a question.

In either of these attacks, the meddler can impersonate a person who is not actually a user of the OSN by creating a fake account on the OSN with that person's information. The same attacks apply even when there is not a "real" asker or askee for the meddler to impersonate, i.e., when the impersonated person does not have an account on the OSN.

In order to maintain shared knowledge secrecy, the meddler must be unable to recover A from the protocol even with an offline dictionary attack. Therefore, any attempt to test whether $G = A$ must require the cooperation of either the askee or the asker, to limit the number of guesses a meddler may make.

3.2.3 Existing Protocols

Two existing protocols satisfy the requirements for our problem. Jablon [36] describes SPEKE, a protocol designed to establish a secure channel between a client and a server who share a common passphrase. As Jablon suggests, this can also be used with shared knowledge as the passphrase between two users. SPEKE is specifically designed to preserve the secrecy and require online verification of the passphrase. The secure channel in SPEKE can be trivially used to exchange a public key once the protocol is complete.

SPEKE achieves these properties by modifying the Diffie-Hellman protocol, replacing the ordinarily fixed primitive base with a primitive base given by a well-chosen function of the shared information. We omit further details of SPEKE.

Ellison [23] describes a multi-question protocol that also satisfies the properties we require. This protocol allows the asker to ask several questions before deciding that she is in fact communicating with the askee. Although it may prevent some honest users from exchanging keys successfully, askers and askees must limit the number of verifications they will perform to reduce the number of guesses a meddler may make.

3.2.4 Embedding SPEKE in an OSN

Facebook is one of many web-based OSNs, and we use it as an example of how one would augment an existing OSN to support SPEKE. Facebook provides private messages between users, which could be used as the communication channel in SPEKE.

Several steps of the protocol require local cryptographic operations that must not reveal certain information such as private keys or the answer. One can perform the SPEKE protocol on an OSN by embedding the protocol in a Firefox extension. SPEKE requires several messages, so the asker and askee must either visit the OSN simultaneously or they must interleave their visits to the OSN several times to complete the exchange. This solution is also appropriate for other OSNs; in particular, Persona [6] already relies on a Firefox extension for cryptographic operations.

In addition to key exchange with SPEKE and exclusive shared knowledge, we can increase a user's view of trusted public keys through a web of trust built on the OSN friend graph. Although users might hesitate to ascribe trust to all of their

Facebook friends, they might be more willing to trust the friends they know well enough to identify through exclusive shared knowledge. We consider the benefit of using a web of trust in Section 3.3.2.2.

3.3 Can Users Ask Good Questions?

Since the security of our system relies on the ability of users to ask good questions, we performed a real-world user study to determine whether users can do so. This study presents a challenge that most user studies do not face: the results depend on getting data about both participants and their friends. Rather than bring individual users in for interviews, we perform our study directly on Facebook to take advantage of the existing friendship information that Facebook provides. We describe our user study, Bond Breaker, in this section.

Like many other viral Facebook applications, Bond Breaker is a social game. We wanted to ensure that users had the right goals while using Bond Breaker, so scoring in the game reflects desirable behavior in an actual system built for secure key exchange. We also believed that a game – rather than a survey – might be seen as fun and might convince users to encourage their friends to participate.

3.3.1 Bond Breaker Game Rules

We present the rules to the users before they begin playing Bond Breaker. In Bond Breaker, users are rewarded for establishing *bonds*. A user establishes a bond by asking a question of a friend, providing an answer, and getting the friend

to provide the same answer; this is analogous to successful (one-way) completion of the key exchange protocol in Section 3.2. Both the asker and the askee are rewarded for successfully establishing a bond, and they are also rewarded for establishing a bond in the other direction.

For example, Alice asks Bob, "Where did we meet for the first time?", and Alice and Bob answer, "a roller disco", forming a bond. Bob independently asks Alice, "What color is my bike?", and both answer "blue", forming another bond in the other direction.

As the name Bond Breaker suggests, we also encourage users to *break* bonds. A user is rewarded when she guesses the correct answer to a question that was not intended for her. A user may break a bond in this way even if the intended askee is unable to answer the question correctly, since this still corresponds to a successful attack in the actual key exchange protocol. A given asker and askee may establish only one bond at any given time: if that bond is broken, they may try to use a new question. Since we want to discourage users from asking and answering poor questions, we penalize the asker and askee whenever a bond is broken.

Continuing our previous example, Eve guesses the answers to Alice and Bob's questions. To the first, she guesses "high school", and fails to break their bond. To the second, she guesses "blue", breaking the bond from Bob to Alice. Unless Eve knows more information about Alice and Bob, Alice's question is good because there are many places to choose from and the answer is relatively obscure. Bob's question is not as good because the answer is easily guessed.

We reward and punish users based on a point system and include a leaderboard

	Asker	Askee	Meddler
Creating a bond	+1 (each)	+1 (each)	-
Breaking a bond	-2 (once)	-1 (once)	+1 (each)

Table 3.1: Scoring in Bond Breaker. Askers and askees earn points for each bond they create and only lose points once per bond if the bond is broken. Each meddler earns points for breaking a bond, even if the bond was already broken by another meddler.

to give users incentive to earn points. We present our scoring rules in Table 3.1. The asker and askee are penalized once for having a bond broken, but arbitrarily many meddlers can earn points for breaking the same bond. We penalize the asker more than the askee when a bond is broken since the asker chose the question and has more at stake in the key exchange protocol; if the bond is established and then broken, the net result would be that the asker loses one point and the askee breaks even.

In our study, each meddler only gets one guess per question, corresponding to the requirement that askers and askees limit the number of answer verifications they will make. In practice, some users asked questions that the askee was unable to answer only because of slight formatting problems and then asked the same question again, giving meddlers an extra chance to answer. We used case-insensitive matching as the only transformation on answers and have not evaluated any other transformations. Any transformation that makes matching more lax will favor usability at the expense of security: the easier it is for friends to identify each other, the easier it will be for an attacker to guess the correct answer.

The rules that we have described provide an effective analogy between success in Bond Breaker and success in an actual system. Users are rewarded for asking

questions of as many users as possible, but punished whenever those questions can be answered by a meddler; in a real system users would obtain benefit from learning many public keys and could incur substantial costs when meddlers convince them to use false public keys. By rewarding users for breaking bonds, we provide an incentive to do so, just as meddlers in an actual system would have incentive to falsify public key information. We believe that Bond Breaker measures well the usability of shared knowledge for identity verification.

3.3.1.1 Data Collection

We opened Bond Breaker to the public on April 3rd, 2009 and collected data for three months. We primarily advertised by word-of-mouth, but also with flyers and a Facebook advertisement. In total, 171 people agreed to participate in Bond Breaker, but 70 of the participants did not ask, answer, or attempt to meddle in any of the questions. Of the remaining 101 active users, 92 chose to ask or answer at least one question while 9 chose only to meddle. In total, there were 225 questions, 200 answers from the askees, and 300 answers from meddlers.

The friend graph among participants in our study is not as densely connected as we would expect in a complete OSN friend graph. 41% of the active users had only one or two friends actively participating in the study. Through user feedback we found that this was a combination of the following: users did not want to bother their friends with what could be seen as spam invitations, the signup process required reading a detailed description of the rules, and users felt discouraged from using the

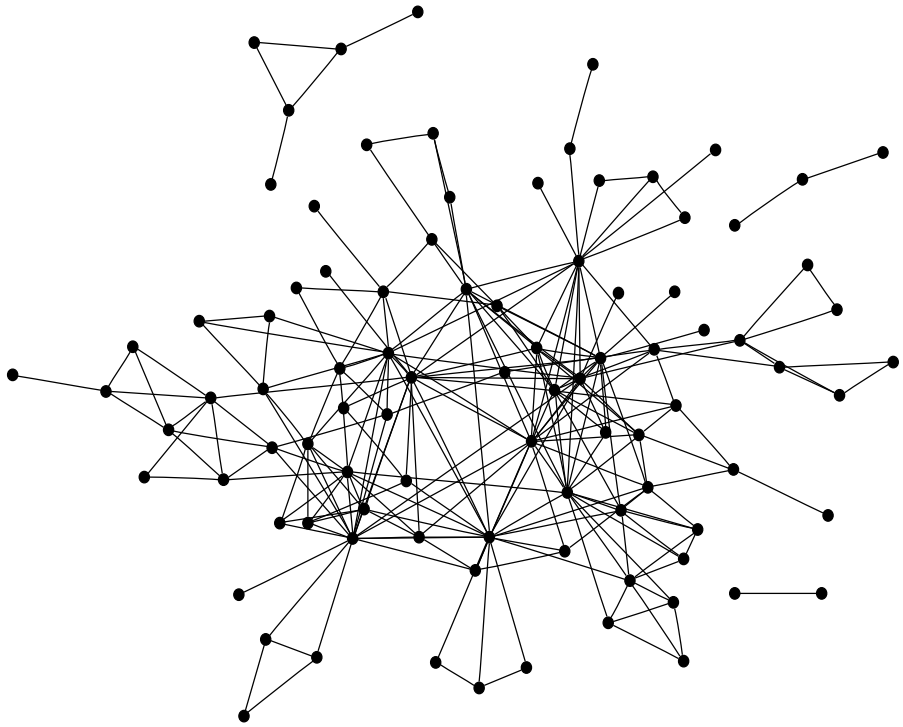


Figure 3.1: Friend graph for active users in Bond Breaker.

application if none of their friends had signed up for it yet. In contrast, 14% of active users had ten or more friends participating; many of these users were connected to each other, forming the densely connected core in Figure 3.1. Most of our results do not depend on how densely connected the friend graph is, but we may be able to obtain more accurate results about the web of trust if we obtain a more complete friend graph in the future.

3.3.2 Results

We use the results of our user study to answer the following questions. Can users easily formulate answerable questions based on exclusive shared knowledge? How likely is an attacker to guess the answer to those questions? Finally, can we

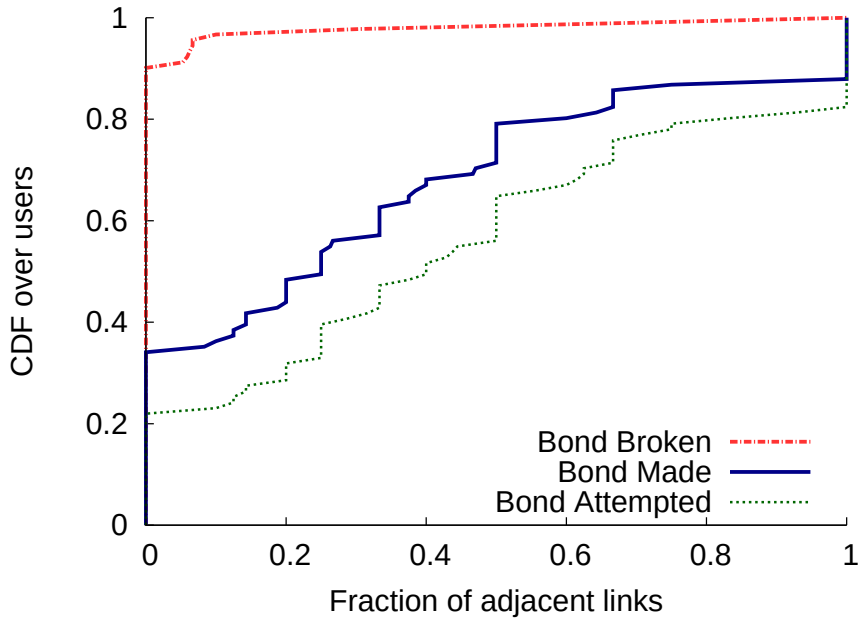


Figure 3.2: Fraction of friends to whom users asked questions, created bonds, and had bonds broken.

use these local verifications to bootstrap a PKI, similar to the PGP web of trust?

3.3.2.1 Question Success Rate

We first consider whether the participants were able to successfully use shared knowledge to establish bonds. Figure 3.2 shows that users had varying degrees of success in their ability to pose questions to friends: about a fifth of the users did not ask questions of any of their friends, another fifth asked questions of all of their friends, and the remainder were distributed nearly evenly. However, when users did pose a question, the friend answered correctly 69% of the time.

Unlike askees, meddlers are rarely able to answer questions, with only a 6% success rate. Table 3.2 shows that strangers meddled with nearly as many questions

	Friend	Stranger	All
Unsuccessful	50%	44%	94%
Successful	5%	1%	6%
All	55%	45%	100%

Table 3.2: Breakdown of meddling attempts based on whether the meddler was a stranger or a friend and whether the meddler was successful or unsuccessful.

as friends. However, five out of six successful break attempts were by a friend of either the asker or askee. Though users may have spiteful friends who try to interfere with their efforts, we expect most attacks in practice to come from strangers. The ratio of successful attacks to attempts is only 9% for friends and 2% for strangers. To provide a point of comparison, web-based security questions can be answered 28% of the time by friends and 17% of the time by strangers [77]. Though this provides a point of comparison to deployed systems, there are significant differences between the problems being solved and the experimental methodologies of these two studies. This comparison is meant only to put the results in context.

From these results we conclude that users are only able to use shared knowledge with some of their friends, and it is usually difficult for a meddler to guess the answer to a shared knowledge question.

3.3.2.2 Web of Trust

Though we have demonstrated that many users only formed bonds with only a small fraction of their friends, we now show that users can learn the keys of other users in the OSN. Figure 3.3 shows the bonds and broken bonds between active users. Based on this graph, we define a user U's *web of trust* to be the set of users

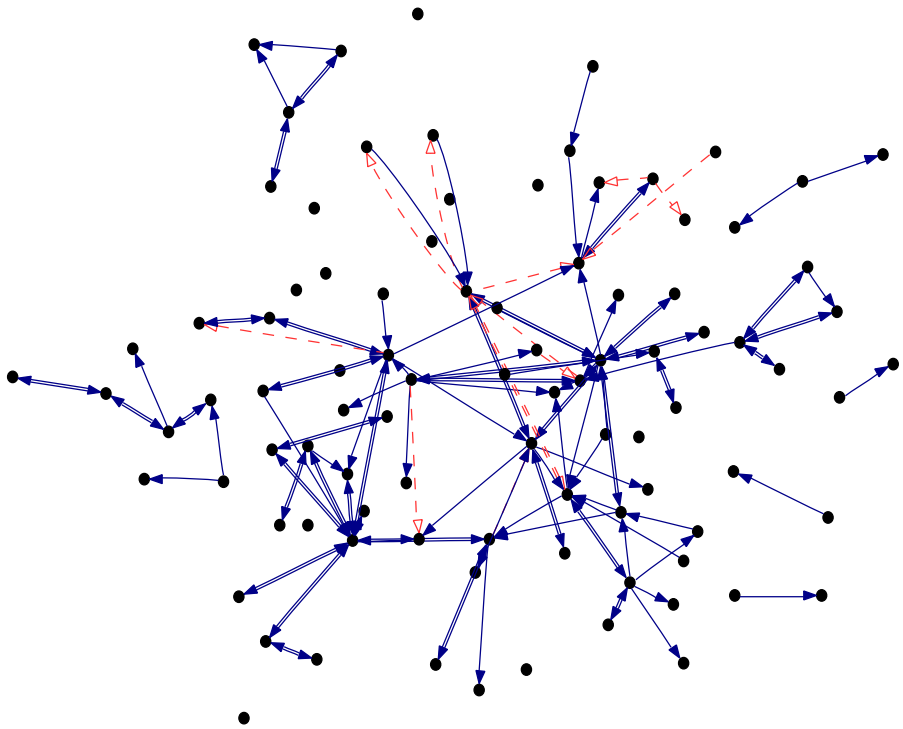


Figure 3.3: Bond graph in Bond Breaker; solid lines with filled arrows represent successful bonds and dashed lines with empty arrows represent broken bonds.

reached by breadth-first search on the directed bond edges beginning at user U. The web of trust may also be limited to a fixed number of hops away from U; a web of trust restricted to 2 hops would include U, any of U's friends to which U has established a bond (hop 1), and any user reachable from those users (hop 2). Restricting the web of trust sacrifices graph coverage for the sake of security. Our definition assumes that trust is related to hops in the friend graph, but in practice we advocate the use of explicit, user-defined trust information.

With a web of trust, the user can do two things: discover the identities of users she does not bond with first-hand, and detect when an attacker has falsified an identity. Since most OSN communication is between friends or between friends-of-friends (FoFs), we focus on learning the keys of those users.

We first show a CDF of the fraction of friends and friends-of-friends reachable via a web of trust in Figure 3.4. If we restrict the web of trust to 2 hops, meaning that the user trusts her friends to attest to keys belonging to her FoFs, 18% of users can identify more than half of their friends or FoFs in the OSN. However, if we do not restrict the web of trust, half of the users with at least one outgoing bond can reach at least 73% of their friends or FoFs. This suggests that the web of trust is a powerful tool in creating a PKI for friends and FoFs and that the study of trust in OSNs deserves further research.

We also discovered that 12 of the 15 unique broken bonds could be detected by the unrestricted web of trust. That is, for 80% of the broken bonds, there is a path of good bonds from the asker to the askee in the unrestricted web of trust. We can use this feature of the web of trust to reduce impersonation in an OSN. Our results

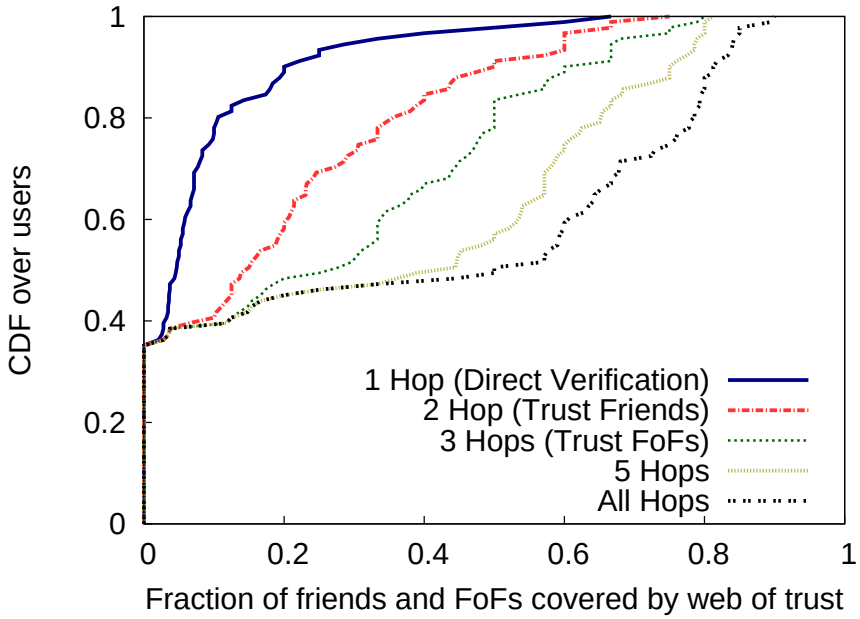


Figure 3.4: CDF of the fraction of friends and FoFs reachable through the web of trust, by web of trust restriction. 36% of the users have no outgoing bonds, so they gain nothing from the web of trust.

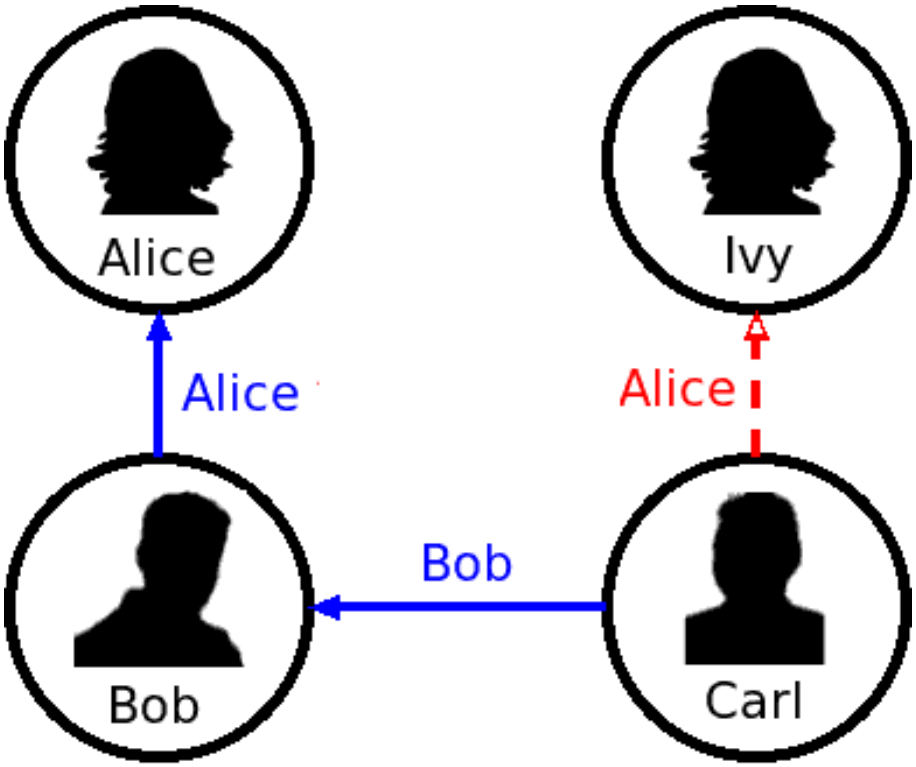


Figure 3.5: Carl incorrectly believes that the account created by Ivy belongs to Alice, and his web of trust does not detect his error.

use Facebook account ids to identify nodes in the web of trust, but an impostor could create a fake account to thwart this. Figure 3.5 shows an example of how the web of trust could fail: the path of good bonds (Carl to Bob to Alice) points to a different node in the friend graph than the broken bond (Carl to Ivy), so Carl will be oblivious to Ivy's attack if he cannot discover that the accounts created by Alice and Ivy refer to the same offline identity. In order to use the web of trust to detect impostors, bond edges must therefore encode information about offline identity to be able to match fake accounts to their real counterparts.

These results demonstrate that we can bootstrap a PKI that provides the keys of friends and FoFs in an OSN. This PKI is distributed and decentralized; we do not require a centralized authority and we can exchange keys entirely online, as opposed to the offline key signing parties of PGP. This PKI is a critical tool for providing security and privacy in emerging OSN-based applications.

Even though shared knowledge is a good starting point for a social PKI, it is not the only technique. We should provide users with as many independent options for in-band key exchange as possible, and shared knowledge is just one of those options. We consider another technique for exchanging keys based on mobile device proximity in Section 4.4.2.

3.4 Conclusion

Impersonation is a fundamental problem of OSNs. We have described how to use exclusive shared knowledge to allow users to take responsibility for identifying

their own friends in an OSN in a completely online way. We described a user study, Bond Breaker, that takes takes advantage of existing Facebook friend information to study our idea in a real setting. We demonstrated through Bond Breaker that exclusive shared knowledge is a practical tool for identifying friends in an OSN and that users can establish a PKI among their friends and friends-of-friends with a web of trust.

Though Bond Breaker reveals the potential of exclusive shared knowledge, it does not test the extent of its use. In both the feedback we received from participating users and from our own experience with the Bond Breaker application, we observed that (1) users have trouble creating questions with difficult-to-guess answers, but (2) users *can* come up with many weak questions that collectively are a thorough test of the askee's shared knowledge. To facilitate the identification of friends, we should take advantage of the weak shared knowledge that users possess more abundantly. We believe that the use of multiple identifying questions may be able to bridge the gap between the results we have presented and a complete PKI, and leave this as a topic of future research.

Chapter 4

Twain

4.1 Introduction

We[1] consider the problem of "rendezvous" over the Internet: two entities wish to communicate but do not know each other's "addresses". More generally, users may want to find entities that possess specific attributes. This general problem appears (and is solved) in many guises, e.g. users of a social network look for other users with the same interest; users wish to construct a secure communication link to another whom they met in the past; peers sharing a file on BitTorrent may need to find peers; Bluetooth devices need to pair before they can communicate; client programs may need to reach a server that is behind a NAT device. Prior solutions have relied on local area broadcast or multicast (e.g, mDNS [16] and Bluetooth [102]), network layer services (e.g., IP anycast [68], GIA [40]), modifications to DNS (e.g., application-layer anycast [10], CDN redirection [20]), trusted-third parties (STUN servers for NAT [74], matchmaking services such as `match.com`), or untrusted well-known servers (location-based rendezvous protocols [55]). While efficient for the domains they were designed for, these ad-hoc solutions do not provide a general solution to rendezvous problems. For example, mDNS, which relies on link-layer multicast, is only useful if two devices are on the same local network; two co-located

[1]This work involved the collaboration of Bobby Bhattacharjee and Matt Lentz.

3G devices cannot discover each other using mDNS. Bluetooth discovery can be used to find nearby resources, the discovery range is limited to a few tens of meters. Existing infrastructure-based solutions are either tailored for network-layer discovery (anycast) or require participants to trust the server (social matchmaking) thereby limiting applicability. We assert that rendezvous is an useful- and common-enough abstraction that it merits being factored out and implemented as a standalone service.

We propose a general *pseudonymous rendezvous* abstraction, called Twain, that can be applied in any of the aforementioned contexts and can be realized as a publicly available service on the Internet. Twain users construct pseudonymous identities and associate attributes that they wish to be discovered by (or query for) with these identities. Using the attributes provided by users, an untrusted third party finds initial matches. In a way, Twain is a formalization of a "process" similar to those employed by matchmaking web sites, where users provide information which is used by the site to generate potential matches. In Twain however, users specify attributes to match on, and then, they may use an interactive protocol brokered by the service to validate that the match before committing to revealing their identities.

Our design goal is to construct a template that can be used for rendezvous in a completely application-agnositic manner. Thus, Twain must efficiently a) capture location context that is implicit in solutions that employ network-layer abstractions (e.g., broadcast [102], link-layer multicast [16]), b) implement diverse matching policies (e.g., location-based matches [18, 55], BitTorrent peer location), and c) preserve user privacy in sensitive contexts (e.g., social matchmaking).

Our primary contribution is an evaluation of Twain by applying the abstraction to seemingly independent applications that span the networking stack and have markedly different requirements and semantics for matches. We show how the Twain template can be used to easily implement rendezvous for BitTorrent clients looking for peers interested in sharing blocks of the same file. However, the Twain framework immediately enables BitTorrent clients to independently construct, advertise, and match on expressive criteria (e.g., peers from a specific AS, peers who support specific transport layer enhancements, peers who offer a particular upload to download ratio) without further protocol change. Using similar ideas, we show how a more general privacy-preserving search application can be implemented in Twain.

We introduce two new applications that benefit from pseudonyous rendezvous: mobile P2P and privacy-preserving matching for wide-area gaming. We show how mobile users can find nearby users who match (essentially) arbitrary criteria regardless of whether the others are on the same provider network or within the same broadcast domain. This application demonstrates that Twain can effectively capture spatio-temporal context, and it is indeed feasible to efficiently implement local search using a wide-area service.

In our game matchmaking application, we describe how proposed systems for matching gamers can leak user location, and describe how Twain can be used to efficiently implement various latency-sensitive matchmaking algorithms. We show how the basic framework is expressive enough to elegantly capture a range of local policies, including policies that restrict revelation of location information only to friends or only to those within a specified distance, and a policy that does not

restrict revelation at all.

The chapter is organized as follows. We provide definitions and describe the design of the Twain abstraction in Section 4.2. We describe a PlanetLab-based implementation and the Twain API in Section 4.3. We enumerate example applications to which Twain can be applied in Section 4.4. We discuss attacks that can be launched by a misbehaving rendezvous service and by malicious users in Section 4.5. We conclude in Section 4.6.

4.2 Pseudonymous Rendezvous

4.2.1 Identities and Pseudonyms

An *identifier* is a globally unique bitstring that names specific data or principal. An *identity* is a public identifier that represents a user[2]. Although we say that identities are public, they may or may not be globally known. We say that information is *bound* to data (such as an identifier) if that information is implicitly revealed when the data is revealed as well. For instance, AS registry information is bound to IP addresses. We say that an identifier is *addressable* if it is possible to deliver a message to the owner of that identifier over some communication channel.

Users possess private information, that is, information whose disclosure they wish to control. Some information is inherently private, for example the private key corresponding to a user's public key, and should never be transmitted over a public

[2]User, in this context, is a participant in a protocol. Users may be human, but could also represent devices or other protocol entities.

communication channel in plaintext. Other information is not inherently private, but the binding between the information and the user's identity is. For example, a phone number by itself is not private information, but the binding of that phone number to a person may be. In this case, transmission of the information over public channels is safe as long as such a transmission does not also reveal the binding.

Pseudonyms are semantic-free opaque identifiers that are owned by a given user. Pseudonyms must provide a proof of ownership, and, by default, should not reveal information that identifies its owner. Therefore, it should be impossible to map a pseudonym to a user, but users should not be able to communicate using pseudonyms they do not own. One way to construct a pseudonym is for the user to generate a public/private keypair. The public key is the pseudonym and the user has the ability to generate signed messages from that pseudonym using the private key; the user can therefore communicate authoritatively as any of her pseudonyms without binding the communicated information to her identity. A user might explicitly bind a pseudonym to an identity, as we describe in Section 4.4, in which case all information bound to the pseudonym is likewise bound to the identity. A user may construct arbitarily many pseudonyms. This is encouraged: we expect users to use independent pseudonyms for different applications so that binding one pseudonym to the user's identity does not affect the user's other pseudonyms.

Pseudonyms, unlike other identifiers, are particularly useful for privacy precisely because, if handled correctly, they are not bound to other identifiers. Contrast this with identifiers typically used for addressing at various layers such as MAC addresses, IP addresses, social network accounts, user-specific public/private keypairs,

and cookie information. These identifiers can be bound to each other — and to users themselves — in a variety of ways [64, 65]. Pseudonyms provide a layer of indirection in which addressing is still possible while binding can be controlled by the pseudonym owner.

4.2.2 Rendezvous Semantics

Twain is a formalization of a "process" similar to those employed by matchmaking web sites, where users provide information which is used by the site to generate potential matches. In Twain, users specify both their attributes and criteria functions for generating a match. Twain provides the following properties.

Asymmetry. Rendezvous is asymmetric: one user makes herself available for rendezvous, subsequently one other completes the rendezvous. Symmetry, if desired, can be achieved if both users make themselves available for rendezvous.

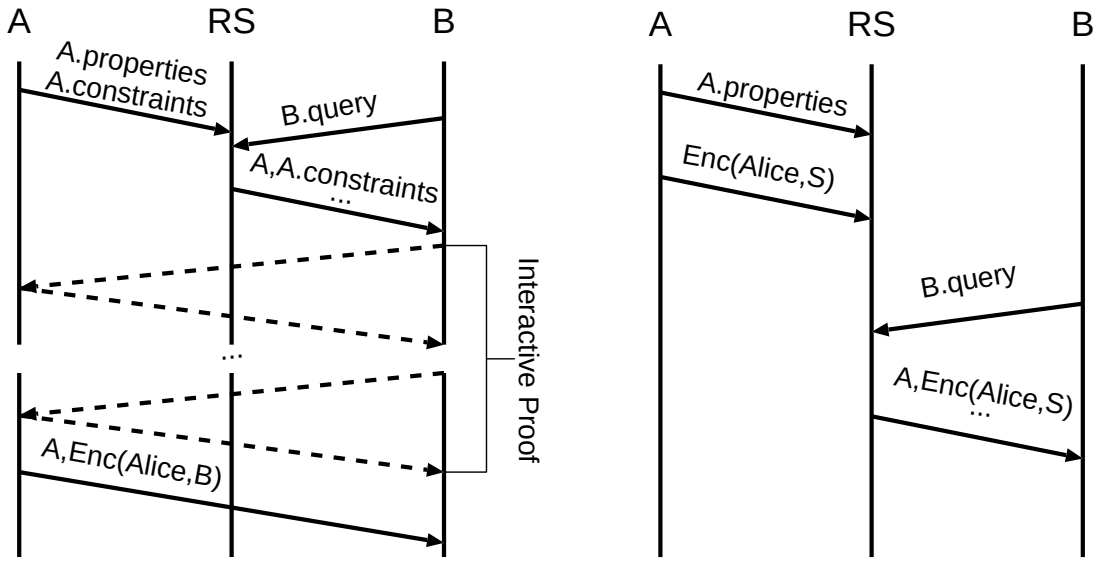
Matchability. Users can rendezvous based on a variety of conditions, including but not limited to: shared information, proximity, and interest based on attributes or tags. A user who makes herself available for rendezvous binds attributes to (only) her pseudonym, and publicly discloses the binding. The user who completes the rendezvous issues queries over published user attributes to locate matching users.

Constrainability. A user who makes herself available for rendezvous maintains control over who may rendezvous with her based on a set of criteria that must be fulfilled by the user completing the rendezvous.

Revelation. When rendezvous completes, at least one of the two users learns the information necessary to continue communication on another channel, typically by binding a pseudonym to an addressable identity. Revelation binds any revealed attributes to both the pseudonym and the addressable identity.

The matchability property provides a lightweight means of filtering out uninteresting users, while the constrainability property provides a heavier, proof-based assurance that the matched users are considered safe to communicate with under the user's policy. Matchability provides scalability, while constrainability provides privacy or security.

We present the overall flow of a pseudonymous rendezvous — match, constrain, and reveal — in Figure 4.1. In this process, one user, Alice, makes herself available for rendezvous under the pseudonym A by contacting the rendezvous service. Then, the other user, Bob, queries the rendezvous service under another pseudonym B, and the rendezvous service returns the matches, including A. At this point, either Alice and Bob engage in an interactive protocol through the rendezvous service so that Bob can prove that he satisfies Alice's constraints, or Alice non-interactively publishes information through the rendezvous service that Bob will only be able to interpret if he satisfies the constraints (through possession of out-of-band information, such as a cryptographic key). Either case concludes with proof that Bob satisfies Alice's constraints and a final message that reveals the binding between Alice and A to Bob.



(a) (b)

Figure 4.1: Alice makes herself available for rendezvous under the pseudonym A, and Bob discovers A through a query. (a) Alice reveals her identity specifically to B after completing an interactive protocol that satisfies her constraints, or (b) Bob learns Alice's identity with local information S that implicitly satisfies Alice's constraints.

4.2.3 Generic Pseudonymous Rendezvous

We introduce the data structures and communication required for a generic pseudonymous rendezvous. All applications that use Twain undertake this basic procedure to generate matches. To illustrate each concept, we use a running example of a customizable BitTorrent tracker. Using Twain to find peers in BitTorrent provides end-users with capabilities that are not part of the base protocol. For instance, the Twain tracker would allow users, for improved performance or for policy reasons, to selectively find peers in specific ASes. The Twain tracker also allows

users to match using non-network related criteria, e.g., a user may specify that she will only peer with others who support specific transport layer features or possess a certificate from a trusted CA or is willing to provide two data blocks in return for each block she uploads. Each of these policies can easily be implemented in isolation; Twain provides a framework for implementing any of them and allows users to craft their own policies as necessary.

4.2.3.1 Definitions

Let RS be the rendezvous service. Let A, B be users. In our example, A makes herself available for rendezvous and B seeks to rendezvous with a matching user (A in this case). Let $\hat{A}$, $\hat{B}$ be pseudonyms for A and B respectively. In this section, we'll assume that the RS executes correctly and returns only (and all) matches that it finds. We discuss different forms of RS and user misbehavior in Section 4.5.

Let A have some properties P_A and some criteria $C_{\hat{A}}$. The properties P_A are a set of bitstrings that represent the complete set of defining characteristics of A. In our BitTorrent example, A may have the property that she belongs to a certain AS and that she is willing to trade blocks of a specific file. The criteria is a predicate that represents the requirements that B must satisfy to learn the binding between A and $\hat{A}$. A may specify a criterion that whomever she peers with must upload two blocks for every block that she provides.

Similarly let B have some properties P_B and some query $Q_{\hat{B}}$. This query is a predicate that is satisfied by the properties of those users with whom B is interested

in rendezvousing. The definitions of A and B implicitly provide the *asymmetry* property we desire. In our example, B wants to peer with a user from a set of whitelisted ASes for the same file, and will be willing to provide twice the upload bandwidth to do so. A's home AS is in the set of ASes that B wants to peer with.

4.2.3.2 Matchability

The Twain protocol proceeds as follows. Initially, A publishes a pair $(m_{\hat{A}}, d_{\hat{A}})$ to the RS. The pair consists of a message $m_{\hat{A}}$ to indicate availability for rendezvous, and a data thunk $d_{\hat{A}}$ that is used for continued processing if a match for the message is found. B publishes only a message $m_{\hat{B}}$ that specifies a query to RS to request matches. If B's query matches A's message, B can use the data thunk obtained via the match to continue communication.

We'll use g to denote a generic matching function that the RS uses to match users. The inputs to g are A's properties, as specified in the message $m_{\hat{A}}$, and B's query (in $m_{\hat{B}}$). These messages are formatted in a standard manner such that the RS can match users without necessarily understanding the underlying sematics of the data being matched. Note also that A may not necessarily want to publish her properties in the clear, nor might B want to publish his query in the clear. Instead, we assume that the properties and query are transformed using application specific functions that obfuscate them as necessary, but still allow matching using the function g. Obviously, the transformation functions are crafted such that such that $g(m_{\hat{A}}, m_{\hat{B}}) = 1$ iff P_A satisfies $Q_{\hat{B}}$.

The output of the transformation functions contains a *key* (k) that is application-specific. Keys are unique for each match domain (e.g., for each application) and allow for multiple matching conditions. Along with the key, each transformation function also emits a value (v) for the match key, and identifies a specific data type (t) and its associated comparator function. Thus, the output of the transformation function is a set of triplets, denoted $m_A = \{(k_i, t_i, v_{i,\hat{A}})\}$ and $m_B = \{(k_i, t_i, v_{i,\hat{B}})\}$. The messages published in this step provide the *matchability* property. For BitTorrent, the keys could be "BitTorrent::FileNameHash" and "BitTorrent::Peer-AS". The data types are string for filename and positive integer for AS, with associated comparator functions string matching and numeric comparison.

We support comparator functions that can match within a range. In general, users may choose to specify upper and lower thresholds that can be used for a non-exact match, or even empty thresholds that indicate that any value will match. Thus, g outputs 1 iff the users agree on the keys being used in the match and the key-wise comparison of *every* tuple lies within both users' specified thresholds.

In our design, the matching function and the comparator functions are known a priori to the RS. Applications transform their input to conform to comparator functions from the set of supported comparators. The set of supported comparators is described in detail in Section 4.2.3.4. The generic matching function g emits the conjunction of all of the individual comparators.

4.2.3.3 Constrainability and Revelation

Upon finding a match, the *RS* returns the matching message and data thunk to the querying user. The data thunk contains sufficient information for the querier to continue. In particular, it may contain a nonce which the querier may use as a key to publish a message and it may contain an encryption key under which to publish information in that message. The message will match a query that the original publisher may issue (or have already placed in the system). Subsequent messages should interactively publish attributes that, collectively, provide a proof that B satisfies A's constraints. Messages published in this step provide the *constrainability* property. This interactive process eventually leads to final message in which one party reveals an addressable identity. The final message provides the *revelation* property.

In our BitTorrent example, the *RS* notifies B of a match, and using information in A's data thunk, B can publish a message with an attribute that asserts that he is willing to upload at least two blocks for each block that A provides. In his data thunk, B publishes a nonce and a public key. Once A submits a query using her original nonce as the key, she will find all matching users, in particular B. A may choose to publish an addressable identifier (IP address and port) encrypted with B's public key using the nonce published by B.

In our implementation, once the *RS* determines a match, it creates an ephemeral session for publishing subsequent messages. Even though each message contains a nonce as described, the *RS* does not have to match over the entire database; it sim-

ply keeps track of sessions with active interaction until the rendezvous is complete or the session times out. Note that the nonces allow the interactive protocol to proceed completely asynchronously — sessions are maintained by the *RS* only to improve runtime efficiency.

4.2.3.4 Data Types and Comparison Functions

To be truly application-agnostic, the *RS* needs to support general comparator functions. However, in order to scale and be implemented as a network-wide service, the comparisons must be computationally efficient.

We implement comparisons over primitive data types, such as integers, floating point values, and strings. We also support comparisons over aggregates, in particular sets and fixed dimensional coordinate spaces.

In the case of arbitrary-sized sets S_1 and S_2, the application can use for $c(S_1, S_2)$ the size of the sets obtained through set arithmetic, namely: $|S_1 \cap S_2|, |S_1 \cup S_2|, |S_1 - S_2|, |S_2 - S_1|,$ and$|(S_1 \cup S_2) - (S_1 \cap S_2)|$. Of these, set intersection is particularly useful for rendezvous since it captures the intuitive notion that two users have something in common.

Enumerating all possible comparison functions on m-dimensional coordinates is difficult. We identify two desirable classes of coordinate comparison functions, aware that there may be useful other classes that we have not considered. The first class is based on the p-norms, namely:

- $\forall p \geq 1$, $c_p(\mathbf{x}, \mathbf{y}) = \sum_{i=0}^{m} |x_i - y_i|^p$

- $c_\infty(\mathbf{x}, \mathbf{y}) = max(\{|x_i - y_i|\}_{i=0}^{m})$

The second class is that of great-circle angular distance on the surface of a sphere. This class is particularly useful for the $m = 2$ case as it provides a way to compare geographical coordinates, a common requirement of location-based rendezvous.

Using these pre-specified comparators is a specific design tradeoff that sacrifices generality for scalability. An alternate design would be to have applications specify an arbitrary matching function, specified in a safe language, that is executed over the properties published by different users. However, we chose to constrain the comparators to sequences of mathematical operations on primitive types as this allows us to introduce the notion of "bounding boxes" that reduce the number of potential matches that the RS has to consider for each user and that assist in partitioning the data in the system. That is, any comparison function c should have a corresponding bounding function $b(c, v) = \{l, u\}$ where l and u represent bounds on the values which could possibly match for the given comparison function and the given value. As we demonstrate next, our choice of comparators provides sufficient flexibility to implement a wide range of rendezvous.

4.3 Implementation

We describe an implementation of the Twain abstraction called TM-1 (Twain, Mark I). TM-1 is a distributed, publicly available service that could for instance be

run on PlanetLab or Amazon EC2. We first describe the TM-1 architecture and then describe the Twain API exported by TM-1.

4.3.1 Architecture

TM-1 is a distributed system comprised of two types of nodes: controllers and data. The data nodes are relational database backends that support our matchability operations for a given partition of the overall rendezvous data. The controllers manage the data partitions dynamically as requests are made, splitting and merging partitions as necessary as data nodes and controllers become overloaded with requests. One of the controllers also serves as a "master" node. The service DNS name should resolve to the master controller, since this node serves as the ingress into the Twain system. Initial rendezvous requests are handled by the the master node, which hands-off the request to a unloaded controller. Subsequent requests in the same session bypass the master. The master node also serves to serialize partitioning operations to maintain consistency in the system. If the master node fails, any of the surviving nodes can take over as the new master. In large deployments, the DNS name could resolve to a load-balancing router, which could direct initial requests to different controllers. We do not use a load-balancing router in our implementation.

The goal of TM-1 is to demonstrate the feasibility of scaling Twain to the needs of real systems. In our implementation, partitioning is done as needed based on the value portion of a data tuple. When a data node becomes overloaded – i.e., it passes

a certain threshold of disk usage – it informs a controller that it is overloaded and picks the splitting value that will most significantly decrease its load. The controller forwards this request to the master who splits the partition and assigns the new partition to the least loaded data node. Queries are then directed to each data node whose partition covers some portion of the query's bounding box as described in Section 4.2.3.4. TM-1's load-based split and merge allows it to scale as match volume increases since new data and controller nodes can be added dynamically as needed.

We evaluate TM-1 on PlanetLab using 20 controllers and 20 data nodes, increasing the number of available and querying clients across experiments from 10 to 110. The clients issue requests immediately after the previous request returns; this is likely to be a much faster rate than we would expect from "real" clients who may only issue one request every so often. The number of clients in our experiments is not meant to directly correspond to some number of users of Twain, but instead indicates the behavior of the system as more clients connect. We present the results in Figure 4.2. The system is able to handle the additional requests as the ratio of clients to servers increases, processing 44 requests per second when there are 110 clients in the largest experiment we ran. Also, the system is able to operate much more efficiently with bounding boxes than without, so it is important for application developers to construct appropriate bounding boxes for applications with custom comparison functions. TM-1 is more of a proof-of-concept than a true test of Internet-scale scalability. We have used it to validate scaling to millions of queries per day using only forty PlanetLab nodes (slivers to be precise). In reality,

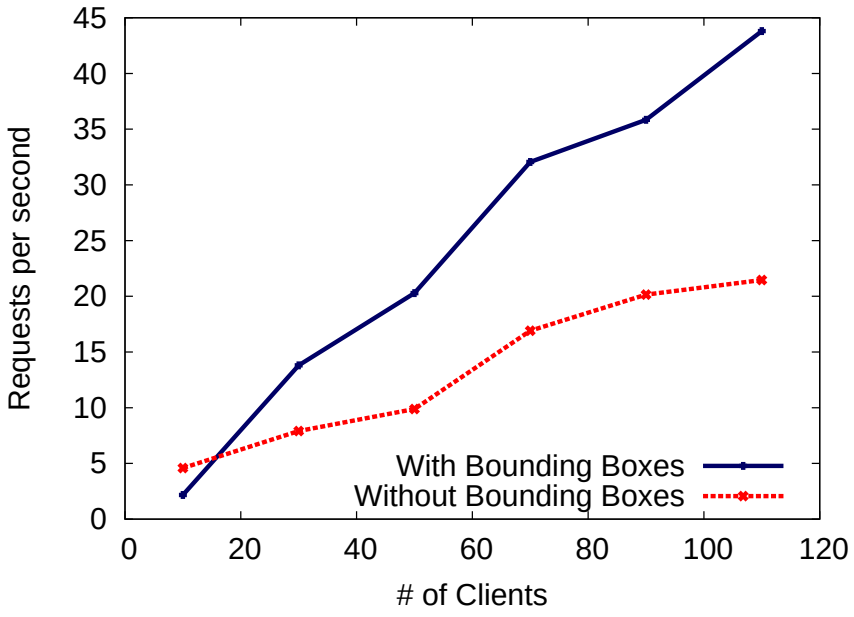


Figure 4.2: Number of requests per second in Twain as the number of requesting clients increases, with and without bounding boxes to assist in partitioning.

we did not (try to) saturate the PlanetLab nodes; instead, we wanted to obtain a conservative measure of the resources required for processing a million queries a day. We believe this deployment is sufficient demonstration that the Twain architecture is scalable.

4.3.2 API

We present the Twain API in Table 4.1. Most operations are associated with a given *service*, such as `BitTorrent`, to assist in data partitioning. Each PK provides a reference for a pseudonym; API calls that take two PK pseudonyms should be thought of as belonging a session between those two pseudonyms, sessions that are established when the `query` function returns non-empty sets of matches. The

Remote functions invoked at a Twain control node	
`available`($service$, PK, $\{(k,t,v)\}$, $expiry$, σ, $channel?$)	$\rightarrow$ $success$
`query`($service$, PK, $\{(k,t,v)\}$, $nonce$, σ, $limit$, $offset$, $store?$, $expiry?$, $channel?$)	$\rightarrow$ $\{PK_i, \{(k,t,v)\}_i, expiry, \sigma_i\}$
`interact`($service$, PK_1, PK_2, $message$, $nonce$, σ, $channel?$)	$\rightarrow$ $success$
`reveal`($service$, PK_1, $PK_2?$, $message$, $nonce$, σ)	$\rightarrow$ $success$
`pollQueries`(PK)	$\rightarrow$ $\{service_i, PK_i\}$
`pollResponse`($service$, PK_1, PK_2)	$\rightarrow$ $(type, message, nonce, \sigma)$
Local functions and event handlers	
`createPseudonym`()	$\rightarrow$ (PK, SK)
`recvQueries`(PK)	$\rightarrow$ $channel$
`recvResponse`($service$, PK_1, PK_2)	$\rightarrow$ $channel$
`onQueries`($channel$)	$\rightarrow$ $\{service_i, PK_i\}$
`onInteract`($channel$)	$\rightarrow$ $(message, nonce, \sigma)$
`onReveal`($channel$)	$\rightarrow$ $(message, nonce, \sigma)$

Table 4.1: The Twain API.

store, *expiry*, and *channel* parameters of the `query` function can be used to provide matching symmetry: rather than have a user issue an `available` call followed by a `query` call, these two operations are consolidated into the `query` call with appropriate parameters.

Under this API, B's query is never implicitly revealed to A: `pollQueries` and `recvQueries` only return results if B initiates an interactive protocol. If A wishes to know B's query, the query can be explicitly revealed as part of the interactive protocol. There is no guarantee that the query that B reveals is the same as the query that B issued to RS, but A can at least verify that her properties satisfy the claimed query.

TM-1 supports both push and pull communication with the rendezvous service. The `recv` functions establish a communication channel that listens for responses from the rendezvous service and triggers the appropriate event handler. Alternatively, the application may choose not to establish an active channel (for instance, if communication is expected to be infrequent and not time-sensitive, or if the anonymity layer between the user and the rendezvous service precludes a persistent response channel) and instead may use the `poll` functions to periodically check for new matches, interactions, or revelations. A combination of these two modes is also reasonable: the application may poll for queries, but once a query is discovered it may want immediate updates for the interactive protocol and revelation to complete the rendezvous as quickly as possible.

4.4 Application Design using Twain

Twain as a standalone service has utility only if it supports a range of applications. In this section, we use examples to demonstrate how application requirements can be mapped on to Twain. We begin with a discussion of the general steps that the application designer must undertake, and follow up with a discussion of five specific application scenarios.

Our goal in this section is to demonstrate the versatility of Twain; we've chosen to showcase the breadth of applications supported by Twain at the cost of a complete analysis of any one.

4.4.1 Process

The process of instantiating an application using Twain follows a sequence of steps —idenfity, map, validate, and reveal— which we briefly discuss in the abstract next.

Identify. The application designer has to identify attributes that she wants to publish or query for, and make a decision about whether these properties and queries should be bound to an addressable identifier. The *RS* is untrusted, thus any sequence of messages that reveal sensitive attributes needs to be anonymized. In these cases, we assume the user communicates with the Twain service using an anonymity layer, such as Tor [21] or a mixnet [15], which serves to decouple attributes from an addressable identity. The Twain API is designed to compose with existing anonymity services.

Map. Once attributes and queries are identified, they need to be mapped to a comparator supported by Twain. However, the problem here is more than molding the application requirement to a supported comparator. The mere existence of a non-obfuscated attribute (or conjunction of attributes) may leak information about a user, regardless of whether the user communicates using an anonymity layer or not. Even for obfuscated attributes, it may be possible for the *RS* or a user to mount a dictionary attack using well-known match keys over small match domains. (For instance, in our BitTorrent example, the a montoring agency will be able to determine whether a particular file is being shared or not simply by iterating over variations of names for the file using "BitTorrent::FileNameHash" key). We describe

possible techniques for addressing the problem of leaked attributes in Section 4.5.

Validate. Once an initial match is found, constraints need to be satisfied using the interactive protocol. The requirements and concerns in this step are similar to the mapping step.

Reveal. Once users validate each other, one of the users must provide an addressable indentifier. As described previously, the interactive communication sequence can be used to securely exchange cryptographic material such that only validated users can map pseudonyms to addresses.

In the rest of this section, we demonstrate how specific applications can use Twain. Our first two examples – mobile peer discovery and privacy preserving game matchmaking – not only demonstrate the utility of the Twain abstraction but also provide novel solutions to unsolved problems.

4.4.2 Mobile P2P: Customizable Local Rendezvous

As smart mobile devices become ubiquitous, mobile P2P systems such as SMILE [55], BlueTorrent [38], and MobiClique [69] rely on the ability of these devices to discover and communicate with other nearby devices for location-dependent purposes. Discovering peers in mobile P2P systems should be much easier even than locating peers in global Internet-based P2P systems because of the inherently narrowed spatial scope coupled with the broadcast nature of wireless communication.

However, locating nearby mobile devices remains difficult. Devices may not always support broadcast (3G) or be on the same broadcast domain (different WiFi

networks). Bluetooth broadcast has limited range, and is prohibited by certain OS/hardware combinations. Even if a robust link layer broadcast were universally available, it is not a panacea for rendezvous. For large user populations, broadcast is inefficient, both at the link layer and because it requires devices to sift through many false positives (imagine using broadcast to find friends at a sporting event). Finally, broadcast solutions inherently leak information because it is possible for third parties to monitor attributes (or queries).

Assuming interested parties can reach a *RS*, Twain provides a potentially elegant solution to the problems we have outlined. The main technical question revolves around whether it is possible to construct robust co-location cues entirely passively. We've designed a mobile P2P application, LoKI, that answers in the affirmative. We next describe how we used Twain in the LoKI design.

LoKI's goal is to allow proximal users to periodically exchange shared secret data for the purposes of post-hoc identity verification in a social PKI. This is similar to Bond Breaker (Chapter 3), except in this case the shared knowledge will be a combination of hard-to-guess bitstrings and the users' knowledge of when social meetings occurred. In order to periodically exchange secret bitstrings, matching users must exchange MAC addresses, a task that we perform using Twain. We assume that users exchange bluetooth addresses, but LoKI is not specific to bluetooth. We designed and evaluated LoKI using commodity hardware, specifically Android smartphones running supported (non-rooted) software.

Map. The relevant attributes for LoKI are the user's location, specific attributes she wishes to be searched on, and her bluetooth MAC address. The user

may also impose validation criteria before disclosing her address. We assume that the user does not wish to share the mapping between her address and her attributes publicly. That is, only other users who are actually nearby and can satisfy all other constraints should be able to see that a given bluetooth MAC address is in that vicinity. To simplify our presentation, we'll describe the case where users wish to locate any nearby user, i.e., the additional attributes and validation are null. These can (and should) be added based on the application- and user-requirements as we have described earlier.

Commodity devices (non-rooted smartphones) limit ambient observations reported to applications. The standard Android API exports a list of visible MAC addresses of WiFi access points and discoverable bluetooth devices and the RSSI for each such signal. Is this information sufficient to find co-located users? In particular, are there sufficient addresses to form a robust cue, and do proximal devices "see" the same addresses? We've conducted a measurement study, described in Appendix A, that suggests that the answer is a qualified "yes" (Figure 4.3). A relatively small match threshold is likely necessary because of the variety of MAC addresses observed even from devices less than a foot apart. However, as we expect most real-world, socially relevant meetings to occur in stationary settings, we believe that rendezvous based on these MAC addresses will generally succeed even with a threshold of, say, $k = 5$ common MAC addresses.

Our current implementation constructs location tags using the information exported by the Android API. This information is susceptible to being spoofed, since the set of visible APs is usually stable. It is possible to construct more robust, time-

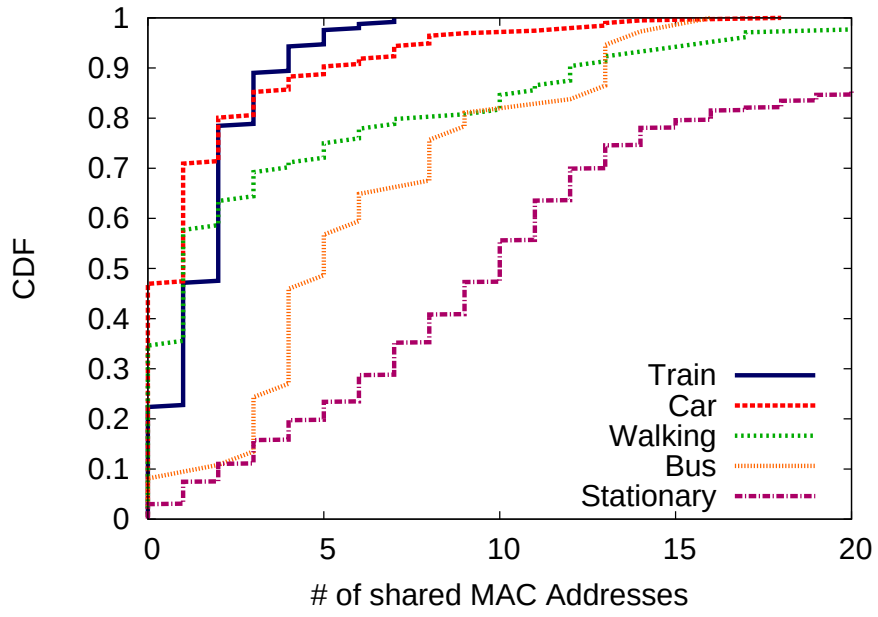


Figure 4.3: Number of common bluetooth and wifi MAC addresses visible from two proximal Android devices at any given time under typical daily movement (described in Appendix A).

varying, signals, for instance by combining the high order bits in the timestamps in each beacon message or with other information [66]. These are not available to LoKI due to API limitations but can readily be incorporated when available.

IP addresses of mobile devices are ephemeral and harder to bind to an identity than non-mobile devices. However, depending on the sensitivity of attributes that are published, even a temporary binding may be undesirable. LoKI connects to the *RS* through Tor, unless the user opts out of this feature.

Map. Let V_A and V_B be the set of visible wifi and bluetooth MAC addresses from A and B's respective devices. Let t be matching parameters that indicate the use of set intersection on strings with a lower threshold of k and an unlimited upper

threshold. Let T be the current application epoch, retrieved from the rendezvous service. A and B publish:

$$m_{\hat{A}} = \{(\texttt{LoKI visible MACs}, t, \{H(M||T)\}_{M \in V_A})\} \tag{4.1}$$

$$m_{\hat{B}} = \{(\texttt{LoKI visible MACs}, t, \{H(M||T)\}_{M \in V_B})\} \tag{4.2}$$

By hashing the visible MAC addresses with the current epoch, neither user reveals the plaintext MAC addresses visible at the location at that time with this published information. Without knowing the plaintext MAC addresses at a location, an attacker cannot track location across epoch boundaries. A also publishes a data thunk that provides the information necessary for the revelation step and an indication that no interactive protocol is necessary.

Validate. In this example, the only constraint that A need put on revealing her identity — in this case, her bluetooth MAC address — is that B knows the threshold number of visible MAC addresses in the vicinity. No interactive protocol is necessary since A does not constrain matches based on any attribute other than location.

Reveal. We reveal A's bluetooth MAC address BM_A through Shamir's Secret Sharing [78]. In this case,

$$e = \{Enc(S(BM_A, M||t, k), M||t)\} \tag{4.3}$$

where $Enc(m, K)$ is symmetric encryption of m with key K and $S(m, p, k)$ computes the secret share for message m, point p, and threshold k. Then, assuming B actually knows sufficiently many visible MAC addresses for the current epoch, d consists of

B decrypting the secret shares corresponding to the MACs he has observed and combining them to recover BM_A. At this point, rendezvous completes with B learning the bluetooth MAC address of A, who both have observed similar ambient MAC addresses and thus are likely to be proximal.

LoKI demonstrates how Twain can be used to implement a MAC-protocol agnostic customizable rendezvous service. Even after a successful rendezvous, LoKI cannot guarantee the establishment of a communication channel; the devices may be barely out of bluetooth range of each other despite being able to observe the same wifi access points. Our experiments show that Twain can be used to replace broadcast for locating nearby users. Twain does not eliminate potential concerns regarding privacy, but we believe LoKI demonstrates that these aspects are orthogonal to rendezvous, even when an untrusted third-party is used to match users.

4.4.3 Game Matchmaking: Privacy-enabled Wide-Area Rendezvous

Users of multiplayer online games are often matched to each other in such a way as to minimize latency, the most critical network measure for a smooth online gameplay experience. Since all-to-all pings are impractical, systems such as Htrae [3] rely on network coordinates [19] to estimate the latency between any two users without testing the connections directly.

Although network coordinates are intended only to estimate latency, work in IP geolocation such as CBG [30] demonstrates that coarse geographical information can be ascertained given latency information from a collection of probes. Since

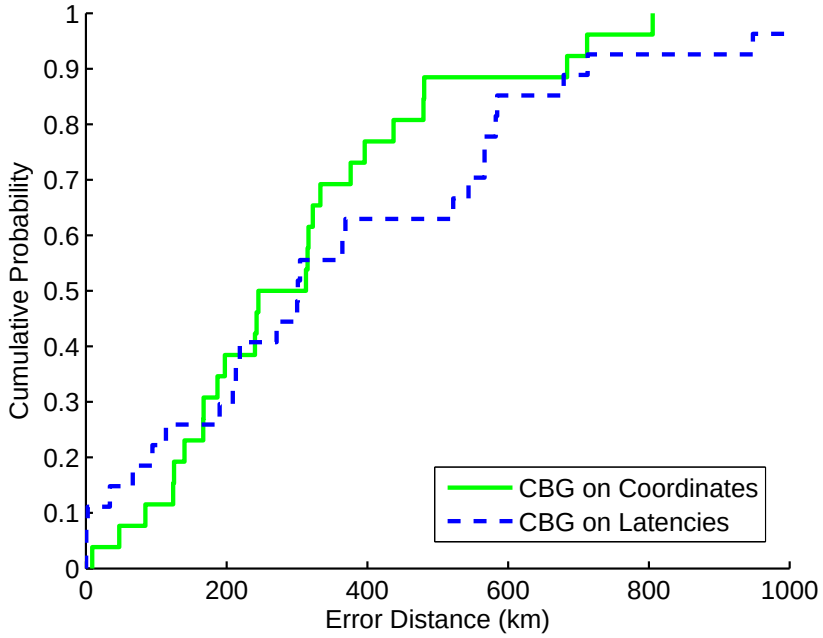


Figure 4.4: Comparison of CBG error distances, defined as the difference between the actual location and CBG estimate, when applied to both measured latencies and network coordinate latency estimates for the East Coast node set.

embedding hosts into a metric space can introduce errors in latency, it is unclear whether CBG can be applied to learn geolocation from network coordinates. We conducted an experiment on PlanetLab to confirm our hypothesis that location can indeed be inferred using latencies gleaned from network coordinates. We describe our experiment and modifications to CBG in Appendix B. Our results, presented in Figure 4.4, indicate that CBG using network coordinate latency estimations performs just as well as when using measured latencies, with a median error distance of 279 km for network coordinate latencies and 301 km for directly measured latencies.

Therefore, network coordinates may be unintentionally and unexpectedly bound to geographical location information. Twain can be used to enhance matchmaking

by providing a measure of privacy for users concerned about leaking location information. We've implemented a matchmaking application, Viveil, that uses Twain to match gamers.

We assume that there exists a trusted authority for a given game, and that this authority provides a certified network coordinate system [39] and certifies that users are in fact valid players (e.g., paying subscribers, not cheaters, etc.). Certified network coordinates are bound to the pseudonyms $\hat{A}$ and $\hat{B}$ upon creation.

The goal in Viveil is not to obscure inevitable direct latency measurements, but to prevent attackers from inferring the geolocation of a user's gamer id from publicly available network coordinates. Said another way, we want to ensure that an attacker must probe the user directly (either with latency measurements to her hard-to-discover IP, or by requesting revelation of her network coordinate) in order to learn the geolocation of the user's gamer id, and that the user should have some measure of control over who may probe her directly.

The actual level of privacy provided will be decided by user policy: some users may not desire unlinkability of geographical location to gamer ids in favor of ease of matchmaking, while others may want stronger guarantees. We identify three policies that represent basic privacy requirements that users might choose: Public, Valid-Users, and Friends-Only. In Public, the user is willing to share any of her properties with anyone, and is only using the rendezvous service for discovery, not privacy. In Valid-Users, the user is willing to reveal her IP address to any valid user who can present a suitably close certified network coordinate, and is only willing to reveal her gamer id to those users who can demonstrate a sufficiently fast

connection for the purposes of gameplay. Finally, in FRIENDS-ONLY, the user is only willing to reveal her gamer id and IP address to friends: this provides much stricter privacy guarantees, though in this case network coordinates are only necessary if the user has many friends.

Identify. The salient properties for a user in Viveil are her network coordinates, her gamer id (such as a Xbox Live gamertag or a Playstation Network ID), the IP address of her gaming device, and a collection of observed latencies to both a) landmarks or peers for the purposes of computing network coordinates, and b) peers from gaming sessions. If the user's IP address is known, an attacker could use standard IP geolocation techniques to estimate the user's geographical location. Therefore Viveil users connect to the rendezvous service through Tor for any policy other than PUBLIC.

Map. Let `G` be a string that represents the game the users want to play, and let l be the maximum latency that that game supports. Let $c_{\hat{A}}$ and $c_{\hat{B}}$ be the network coordinates of A and B. Let t be the type of network coordinate along with the coordinate comparison function[3] and with no lower threshold and an upper threshold of l milliseconds. Then A and B publish $m_{\hat{A}} = \{(\texttt{Viveil:G}, t, c_{\hat{A}})\}$ and $m_{\hat{B}} = \{(\texttt{Viveil:G}, t, c_{\hat{B}})\}$, respectively.

Validate. No interactive protocol is necessary for the PUBLIC or the FRIENDS-ONLY policies, so A's data thunk will reveal her IP address as described in the next step.

[3]To our knowledge, our implementation supports comparisons of any type of network coordinate proposed in the literature through the generic mathematical expression comparator.

For the VALID-USERS policy, A publishes a data thunk that instructs B to publish his IP address (suitably encrypted), which he does. A first verifies that the coordinate in $m_{\hat{B}}$ is a valid certified coordinate and that it lies within the acceptable threshold l, then she likewise publishes her encrypted IP address. These IP addresses are twice encrypted: first, with the public key associated with the other user's pseudonym for one-on-one communication, and second with a symmetric key periodically distributed to all users considered to be valid users by the game authority. At this point, A and B can connect directly over the Internet and measure the latency between their gaming devices. A decides to reveal her identity (her gamertag) after comparing this measurement to l. Note that this requires that both A and B subscribe to the VALID-USERS policy; if A does but B does not, he cannot rendezvous with A since he is unwilling to reveal his IP address.

Reveal. Under the PUBLIC policy, e is simply the plaintext gamer id and IP address information, and d is the identity function.

For the VALID-USERS policy, A encrypts her IP with $\hat{B}$'s public key, and B simply decrypts with his private key.

For the FRIENDS-ONLY policy, B's friends distribute nonces periodically (say, hourly). A publishes the nonce she received from B hashed with $\hat{A}$ in plaintext and B compares the published nonce to the list of received friend nonces.

4.4.4 Privacy Preserving Search

Search describes many social networks applications. By aggregating information through a service, users can find: classmates or peers (LinkedIn, Facebook), buyers or sellers of a product (Craigslist, eBay), potential dates (Match.com, eHarmony), tagged content (Twitter, Facebook, Blogger), or nearby friends, strangers, or content (Google+, Foursquare, Yelp).

Often this information is private in nature, so publishing it to an untrusted service raises privacy concerns for users. Application-specific solutions exist [42, 54, 66, 80] to handle search in its various forms, and many of these solutions can be straightforwardly applied using our rendezvous service as the aggregation point. For certain user policies the indirection provided by pseudonyms can further enhance the user's privacy guarantees beyond the original solutions.

Policies. The relevant properties of users in a privacy preserving search application vary based on what is being searched for, but generally can be expressed as a collection of key-value mappings where the keys are strings and the values are either strings or geographical locations. Some individual key-value pairs are inherently private while others are only considered private if used in conjunction with other key-value pairs.

For example, a user interested in selling a used car might have the following properties: {`product` : 'car', `make` : 'Honda', `model` : 'Civic', `year` : '2009', `color` : 'orange', `price` : '$18,000', `location` : 'City, State', `email` : 'user@host.domain', `phone` : '800-555-1212'}. Individual values such as the user's email address and phone number

— values which are not unlike identities in their own right — may be considered to be completely private, intended only to be revealed after a successful rendezvous (i.e., after the rendezvousing user satisfies the criteria set forth by the seller). Though other individual values may not be private, the combination of them may be able to uniquely identify a user[4].

There exists a fundamental tradeoff between findability and privacy that can only be reconciled by the user's privacy policy, though this policy can be informed by querying the state of the system. Before making the user available for rendezvous, the application can test queries over the properties the user is publishing to see if there are sufficiently many other users, providing a sort of empirical k-anonymity [97], advising the user to, for instance, remove the car's color if she does not want it to be uniquely identifiable. Recent work [91] has tried to quantify privacy leakage as more information is revealed, and a search application could use these techniques to better inform users.

Without a verifiable proof of possession, however, attackers can create fake products to pollute attribute statistics. Sampling, dictionary, and pollution attacks are inherent in any system that aggregates user information. Twain does not introduce a new mechanism for either proving possession of attributes or for addressing sampling attacks, but it does not enable new attack surfaces either.

Since we expect users to have an interest in at least some amount of privacy

[4]On Jan 27th, 2012, within 125 miles of the authors, there were over 5000 Honda Civics listed for sale on the autotrader.com web site. Only two were orange. There were no orange Honda Civics available within 100 miles.

and it is not easy to predict the linkability of a user's IP address and arbitrary searchable information, we conservatively use Tor as a default anonymity layer for this application.

Map. For the purposes of deciding how to transform her properties prior to publishing, we first must know: is the user making herself searchable to friends or to strangers? For simplicity, we consider a friend to be anyone with whom the user shares a piece of secret information, such as a symmetric key, while strangers lack this information.

To be searchable by strangers, the user must advertise plaintext versions of her properties according to her policy decisions. If the user is concerned that some combination of her properties will identify her too narrowly, she splits the properties into subsets across multiple pseudonyms $\hat{A}_1$, $\hat{A}_2$, ..., $\hat{A}_n$ and publishes these properties as $m_{\hat{A}_1}$, $m_{\hat{A}_2}$, ..., $m_{\hat{A}_n}$.

There are two alternative options for being searchable by friends based on obfuscated versions of her properties. These options differ in type of overhead they introduce, making them appropriate for different classes of applications. In both cases, we assume that the user periodically distributes a temporary symmetric key SK_A to all of her friends. This scheme applies equally well to attribute-based groups as used in Persona [6].

In the PUBLISH-PER-FRIEND case, A creates a pseudonym $\hat{A}_{f_i}$ per friend and publishes values under those respective pseudonyms that contain $Enc(P_A, SK_{f_i})$, i.e., she encrypts her data independently for each friend. Then B can simply match once with the value $Enc(Q_B, SK_B)$ to find all of his friends' content that matches

the given query. This option is more suited to applications where there are fewer publishes than queries, such as user profiles or product sales.

In the QUERY-PER-FRIEND case, A creates a single pseudonym $\hat{A}$ and publishes values that contain $Enc(P_A, SK_A)$. B on the other hand, issues queries under pseudonyms $\hat{B}_{f_i}$ containing the value $Enc(Q_B, SK_{f_i})$. This option is more suited to applications where there is much more content published than there will be queries, such as searching through all of the status updates on a friend feed.

Validate. Users of privacy-preserving search have many options for interactive protocols that can be used to constrain who may learn their identity, depending on the nature of the data being searched:

- Shared knowledge questions, as in Bond Breaker (Chapter 3).
- CAPTCHAs [89] to thwart automated identity harvesting.
- Biometric tests. For instance, A could request that B include an audio clip of a certain phrase, if A would recognize B's voice.
- Reputation-based certifications through a trusted third-party service, for instance to ensure a highly-rated buyer on eBay without necessarily revealing identity.
- Currency exchange, perhaps with BitCoins [61].
- Information from other distributed credit or reputation systems, such as Credo [87].

Reveal. Upon successful completion of the protocol in the previous step, A reveals her identity encrypted with $\hat{B}$'s public key.

4.4.5 NAT Traversal using Twain

A well-known problem stems from the use of Network Address Translation boxes (NATs) designed to address the limited number of available IP addresses on the public Internet. A device behind a NAT can easily communicate to other devices with public IP addresses, but not to other devices behind NATs. A simple solution to this is STUN [74], in which one of the devices contacts a STUN server to learn the public-facing IP address and port allocated by the NAT, advertising that addressing information to the other user. This assumes that the users have some way of addressing each other at a higher layer. Twain naturally provides both higher-layer addressing in the form of rendezvous matches, and can easily provide the public-facing IP address and port to any host that contacts it, even if that addressing information corresponds to the egress of an anonymity layer.

4.4.6 BitTorrent and other P2P Applications

We have already described an enhanced BitTorrent tracker built using the pseudonymous rendezvous abstraction in Section 4.2.3. The same model can be applied for other decentralized peer-to-peer applications as well. Clients for decentralized multi-user applications, e.g., DHTs [73], media streaming [90], backup [49], can use the same schematic to replace ad-join protocols. As a bonus, using Twain will allow users to specify customizable join criteria, and also maintain privacy if users choose to join using a anonymizing layer.

4.5 Discussion

Users access a rendezvous service (RS), which is an untrusted process accessed over the Internet at an publicly known address. Minimally, users expect the RS to store attributes and queries, and produce matches within a reasonable time period.

We assume that the RS can reveal messages it receives to other parties (effectively rendering it a public communication channel). The rendezvous service could also return false rendezvous matches or false messages from a pseudonym it does not own, but this type of misbehavior can be detected by the user. The RS may suppress matches; without external protocol mechanisms, such as randomized checks or witness sets [33], match supression cannot be detected by users.

4.5.1 Matching over Sensitive Data

Users need not entrust the rendezvous service with private data. However, some applications, e.g., a service matching potential dates, may entail the rendezvous service matching pseudonyms using attributes that contain sensitive information. Even though a match reveals only a pseudonym, existing work has demonstrated that relatively small amounts of correlated information [84, 64, 37] is sufficient to identify individuals.

Twain does not provide a generic solution to deanonymization, and the rendezvous service can mount known attacks to try to unmask pseudonyms. The restricted case in which users wish to rendezvous with others with whom they've previously shared a secret key has been addressed using identity-based cryptogra-

phy [63]. In this setting, the rendezvous service does not learn any information about the attributes over which users are matched. The general case is open: a possible approach is for users to intentionally create "Sybils" – a set of plausible user profiles along with their own and submit each profile with a different pseudonym. The Sybil profiles should be created using expected distributions in each profile category such that the rendezvous service (or any other users) are unable to distinguish Sybils from "real" users. This scheme assumes that the entire match database is public, enabling users to able to pick attributes using profile value distributions. A similar approach for obfuscating real user information using Sybils generated via the expected distribution of attributes is described in NOYB [31]. Note that if the database is not public (or when the system is being initiated), users may choose Sybil attributes that follow the distribution of public profiles in existing social networks, as described in NYOB.

Sybil profiles would (ideally) ensure that the rendezvous service cannot ascertain whether they are matching a "real" user or a Sybil. In the worst case, as long as real users choose attributes carefully (by following a known distribution), they can plausibly deny being the owner of a pseudonym that matches their known attributes. Further, the availability of the expected attribute data provides the user (or application) with sufficient information to obtain a measure of the amount of identifying information that is leaked by information associated with a pseudonym. We note that the creation of Sybil profiles that mixes user data within noise for placement in a untrusted database is a dual of well-known ideas in differential privacy [22], whereby a trusted database perturbs query results with noise such that

untrusted users cannot ascertain details of individual datum.

4.5.2 User attacks: Denial-of-Service

In addition to attacks by the *RS*, it is worth noting at this point that users are generally untrusted as well. The rendezvous service cannot distinguish users; it deals purely in pseudonyms and so a user may issue arbitrarily many queries under different pseudonyms. Malicious users can therefore launch various attacks. User initiated attacks fall into two broad categories: users may attack the system by launching a DoS attack consisting of spurious match requests. Alternately, malicious (or curious) users may try to deanonymize pseudonyms by launching "dictionary" attacks that scan through different attributes.

Even though users are anonymous, the rendezvous service can address DoS attacks by using well-known techniques such as CAPTCHAs [89] or automated proof-of-work techniques described in Portcullis [67]. Dictionary attacks by users (in the worst case where users are not rate-limited or if the *RS* colludes with users) are equivalent to deanonymization attacks launched by the rendezvous service, and are addressed in the same way.

4.6 Conclusion

We have described the Twain abstraction, a primitive that allows two users who wish to communicate but who do not know how to address each other to find each other based on potentially private information. We provided diverse examples of how

the abstraction could be used to solve problems in networked systems, demonstrating the utility of the abstraction as a modular component in system design. We have shown how Twain can implement well-known rendezvous mechanisms, and how re-casting old problems (such as BitTorrent peer-selection) within Twain enables more expressive solutions. We have described two new problems —mobile P2P rendezvous and privacy-preserving matchmaking for games— for which Twain provides clean solutions. Finally, we have described an implementation of Twain and demonstrated its feasibility for use in actual systems.

Chapter 5

Related Work

5.1 OSNs

5.1.1 Privacy Leakage

Several works examine the characteristics and recent growth of OSNs [27, 43, 45, 59, 60]. Krishnamurthy and Willis [47] study how OSNs share users' personal data with third parties such as applications and advertisers. They note that Facebook places no restrictions on the data that is shared with external applications. Advertisers use personal data, as well as information acquired through cookies, to serve targeted ads.

Prior research has characterized privacy problems with OSNs. Acquisti and Gross [1, 29] show that Facebook users at CMU often share more data than they are aware of. Lam et al. [48] study a Taiwanese OSN to show that users' annotations compromise the privacy of others. Ahern et al. [4] study Flickr to see how location information is leaked through users' photographs. Several studies [34, 44, 93] exploit the friend graph to infer characteristics about users. Persona resolves these issues by allowing users to precisely express the policies under which their data, including friend information, is encrypted and stored.

5.1.2 Privacy-enabled OSNs

The research community has recognized the problem of privacy in OSNs and proposed several solutions which build on top of existing OSNs. NOYB [31] hides an OSN user's personal data by swapping it with data "atoms" of other OSN users. NOYB provides a way to map these atoms to their original contents. flyByNight [52] is a Facebook application that facilitates secure one-to-one and one-to-many messages between users. Finally, Lockr [86] uses ACLs based on social attestations of the relationship between two users, similar to how Persona distributes *ASK*s to users that satisfy certain attributes. Persona and Lockr both use XML-based formats to transfer privacy-protecting structures.

5.1.3 Access control and ABE

In Persona, the attributes a user has determines what data they can access. This resembles role-based access control [26] and attribute-based access control, which bases authorization decisions on the attributes assigned to users [12, 96]. Attribute based encryption (ABE) was introduced as an application of a type of identity based encryption (IBE) called fuzzy IBE [75]. Unlike early ABE schemes, CP-ABE [9], which Persona uses, binds ciphertexts to access structures while secret keys contain attributes. Ciphertexts can be decrypted with a key that contains a set of attributes that satisfies the access structure. Multi-authority ABE [14, 51] removes the need for transitive key translations but requires each user to have a globally-unique identifier and the attribute set to be partitioned amongst the users.

Pirretti et al. [70] show how to build a dating social network that only reveals information about a user if their attributes match another user's desired description. Unlike Twain or Persona, their system relies on a single authority to generate all secret keys. Traynor et al. [88] introduce a tiered architecture to improve the performance of ABE so that it scales to millions of users.

5.1.4 OSN Impersonation

Toomim et al. [85] show that shared knowledge could be used as an alternative to group-based access control in OSNs. In their work, users protect OSN content by guarding it with a question; only users who can answer the question can access the content. In contrast, Bond Breaker uses *exclusive* shared knowledge to verify identity rather than group membership, and exchange cryptographic keys on which access control can be built.

Studies of OSN security show that current methods of identifying users are insufficient. Bilge et al. [11] describe an attack in which the attacker copies a victim's information from one OSN to another to impersonate the victim on the new OSN. This allows the attacker to befriend the victim's friends, learn information about them, and continue the attack on those new users. Key exchange with shared knowledge could be used to prevent such attacks; the attacker may have access to personal information, but not exclusive shared knowledge. Felt [25] also describes an exploit for hijacking Facebook accounts (which has since been patched). Our work could be used to detect hijackings and repair them.

Alexander et al. [5] describe a modification to off-the-record (OTR) instant messaging that allows users to authenticate each other using shared secrets. Stedman et al. [82] study how a small group of users interact with the modification. Our work instead considers shared knowledge in the broader arena of OSNs and quantitatively demonstrates the ability of users to employ shared knowledge.

5.2 Rendezvous

5.2.1 Local Area Rendezvous

Multicast DNS (mDNS) [16] allows networked devices to locate services in the local area using a link-local top-level domain (".local") . mDNS uses link-layer multicast to advertise and query for resources. Resource information is stored using DNS records of type SRV (service) or special TXT records known as DNS-SD (service discovery), also known as "Rendezvous" records. Unlike Twain, mDNS requires participants to be on the same broadcast domain (link), and requires users to explicitly advertise services on the local network.

Bluetooth [102] devices broadcast query messages in order to find nearby devices, with those in discoverable mode responding to the query, supplying device properties and information on its addressable identity (MAC address). Through the process of pairing, which completes the rendezvous between two devices, both devices generate a link key in order to create an encrypted channel for further communication.

By design, Bluetooth is only useful for pairing users within range (few tens

of meters), and both Bluetooth and mDNS rely on a broadcast channel. Unlike these protocols, Twain allows users on a local network (or nearby users regardless of network) to rendezvous without divulging interests or queries on the local network. We further discuss the merits of non-broadcast rendezvous in Section 4.4.2.

5.2.2 Location-Based Rendezvous

SMILE [55] is a protocol for anonymous communication between users who have encountered each other at some time in the past. SmokeScreen [18] provides similar functionality, including "presence-sharing" among users. In both systems, devices exchange cryptographic information in the background, which can subsequently be used to initiate a conversation. Both systems use a server (untrusted in SMILE, trusted in Smokescreen) for rendezvous. These protocols are prominent examples of how the pseudonymous rendezvous abstraction is being replicated ad hoc on a per-application basis.

Detecting proximity based on observable, location-based signals is a well-studied problem with a number of solutions for other settings. Narayanan et al. [66] describe the use of location tags to allow for private proximity testing of two devices, for instance if two devices with known identities want to verify proximity even though they do not intend to communicate. By relying on wifi broadcast packets for location tags, general non-rooted devices will not be able to participate in their system. They also assume the existence of known identities — the very problem we are trying to solve.

Varshavsky et al. [57] and Mathur et al. [58] similarly describe mechanisms for confirming device identity based on observable radio signals to prevent man-in-the-middle attacks on Diffie-Helman key exchange. Their goal is slightly different than ours: they wish to ensure that two specific devices pair securely based on proximity, while we wish to construct temporary pseudo-anonymous identities for all proximal devices that can be matched to permanent identities later.

Applications such as KeySlinger [111] and Bump [103] use real world meetings to exchange information, including public keys. These exchanges are user-initiated at the time of the meeting, but we provide flexibility for users to initiate key-exchange long after a meeting has occurred, which we believe more closely resembles typical establishment of OSN relationships.

5.2.3 Pseudonymous Communication

In his Ph.D. thesis, Goldberg [28] describes PIP, a pseudonymous communication infrastructure for the Internet. Many of the ideas he presents have influenced our work, though there are key differences between PIP and Twain. Most notably, PIP allows for pseudonymous rendezvous but it requires that one end of the rendezvous have a well-known identifier, restricting PIP to only client-server rendezvous. I3 (Internet Indirection Infrastructure) [83] allows for pseudonymous rendezvous-based communication, where the pseudonyms of the users exist as trigger id's in the system. In order to send packets, senders are only required to know the corresponding id for the receiver, without needing to use the actual address-

able identity (IP address) of the receiver. By matching on more complex criteria, Twain supports peer-to-peer rendezvous, enabling more applications including the ones described in Section 4.4.1.

Unlike PIP and I3, Twain is intended only as a communication bootstrapping mechanism: once two users rendezvous, they are expected to continue their communication elsewhere. If this intention can be enforced, e.g., using the techniques outlined in Section 4.5, it will limit both the resources that Twain must provide and the ability of malicious users to abuse the service for illicit distribution. This property of Twain also enables it to be a globally available persistent service as opposed to the ad hoc, temporary services described by Goldberg.

Finally, there is a fundamental difference in how pseudonyms are meant to be used in Twain, compared with that of PIP and I3. In Twain, the pseudonyms are a means to an end; once rendezvous has completed, we expect that at least one of the two parties involved will bind her pseudonym to a real identifier. Twain, unlike PIP and I3, is not specifically intended for long-term pseudonymous communication. Despite this intention, Twain does support pseudonymous communication; the user need only bind her temporary Twain pseudonym with some other addressable pseudonym at the end of rendezvous.

5.2.4 Wide-area Resource Location

RFC 1546 [68] (Host Anycasting Service) is perhaps the earliest wide-area attribute-based routing protocol defined for IP. Hosts share the same anycast IP

address, and packets may be directed to any host that shares the address. Thus, it is possible to rendezvous using anycasting using only the anycast address, instead of the unique IP address of the destination host. Global IP-Anycast [40] describes a scheme for scalable routing of anycast addresses. The idea of rendezvous is the same, with anycast hosts being reachable without needing to know their unique IP address. Anycasting is routinely used to provide fault-tolerance and improve performance for wide-area distributed services, prominently the root name servers [76]. Being an entirely network layer abstraction, anycasting is limited in the flexibility it permits in terms of "attributes" that a host can advertise, and how queries are matched to hosts.

Anycasting can be implemented at higher layers as well: Application-Layer Anycast [10] is an extension to DNS that uses a top-level domain ".any" to allow clients to specify metrics which are used to select one from a set of eligible servers. Application-layer anycasting is more flexible than network-layer implementations, but in that it is designed to resolve hostnames. Intentional Naming Service (INS [2]) uses attribute/value names for message routing. Along with regular name resolution, INS supports late-binding anycast and multicast. INS supports message forwarding based on application-specific metrics for early-binding and anycast. Like these systems, Twain can be used to resolve hostnames, but allows for more flexible matching criteria, including approximate matches and interactive proofs, which would be likely be cumbersome to encode in a hostname.

5.2.5 Publish-Subscribe Protocols

Publish-subscribe protocols [8, 13, 24, 83] enable asynchronous communication and rendezvous by design. Twain is akin to a publish-subscribe system to which privacy conscious users connect using an anonymous communication system. However, Twain provides a structure for communication that is tailored for locating resources by attribute, and not for continual receiving continual messages on subscribed channels. Unlike most pub-sub systems, Twain allows not only the querying user, but also the the advertising user to constrain matches. Users may specify policies, in the form of an interactive protocol, that are used to validate that both advertiser and querier possess requisite attributes.

Chapter 6

Discussion

6.1 Open Problems

Many challenges remain in building systems that allow users to manipulate private data through the types of applications to which they have become accustomed. Though I have provided technical solutions to many of these problems, I anticipate that future research will yield novel solutions to the problems I have yet to solve.

Although I have described two very broad classes of applications – content-agnostic applications and applications that operate on semi-private data – there are applications that do not fit into either of these categories. That is, there are applications that require an unobfuscated mapping between user identities and data contents. Future work in this area would be to first, carve out any more classes of applications that can provide privacy without trusting a third party; second, prove that any remaining applications cannot provide privacy without trusting a third party; and last, for a given application thought to lie in this class of remaining applications, search for alternative means to provide the application service that would allow the application to be included in one of the other classes.

One example for another class of applications that might include applications not already found in the two classes I identify in this work is the set of applications

that can provide privacy through secure multi-party communication techniques [94], i.e., when two or more known parties want to agree to the output of some function of their private attributes. Developing a rigorous taxonomy of privacy-enabled applications – along with the intersections between application classes – is a natural extension of the work I have started.

Much of my work includes proof-of-concept implementations that demonstrate that a given design is practical, but they do not consider the many optimizations that could be applied to my designs or what the privacy implications might be for those optimizations. A more rigorous evaluation of my designs would be a significant contribution, but has so far been hampered by a lack of extensive, realistic benchmarks on which to test them. One future venture that I would strongly support is that of a social network simulator, to provide the power and convenience of network simulators to the social network setting for system designers. I envision that a social network simulator would provide empirically-derived, configurable settings to simulate both the friend graph and application workloads for core OSN services such as the activity feed, wall, profiles, and photos. The system designer would define simulation parameters to choose the size of the network, how densely connected its users are, and a distribution for the endpoints and sizes of data transmissions across the social network; ideally user-to-user connections and data distributions could be derived empirically from real networks. A social network simulator would facilitate evaluation of many ideas including differential privacy, scalable OSN data storage architectures, and social data caching.

Though I provide high-level descriptions of the privacy concerns associated

with certain applications and sketch solutions to provide certain privacy policies, individual attention should be applied to these problems to construct complete and rigorous privacy solutions. This is especially true for those problems that I have newly identified, namely, mobile peer-to-peer background communication and privacy-enabled game matchmaking.

Revocation of access to data remains a challenging problem; once data has been published, it is nearly impossible to verifiably destroy it, and moving forward with correct privacy requires expensive, difficult, and complicated key distribution mechanisms. Technical mechanisms that assist a user in recovering from a privacy error are unfortunately lacking and would be a beneficial addition to the advocacy of user-oriented privacy design.

6.2 Deployment Hurdles

One of the hardest questions that remains is how to migrate users from existing social networks to a newly designed one. The technical solutions that I have described provide certain objective advantages over state-of-the-art practices in current OSNs, yet current OSNs possess a sort of "gravity" that discourages competition from new OSNs. That is, the more users an OSN has, the more utility that OSN can offer to a typical user. Persona, if fully developed with many Twain-based applications, would still have difficulty getting off the ground without a substantial initial influx of highly-connected users. Though this is not a technical problem, there may be more technical contributions to be made to facilitate the network effect [41] on

a new OSN.

Another question is whether users actually want to be in control of their private data, and if so, how much are they willing to pay for it? Certainly there are some users who do care, but if the majority of users do not care about privacy, will it even be possible to entice enough users to use an OSN in which the only advantage is that of greater privacy control? Here I am somewhat discouraged: history suggests that ease-of-use is far more important than privacy as users have transitioned to cloud-based storage [104], and membership of pay-to-use services is generally less than that of comparable advertising-based free services [107]. If I realistically expect for *most* users to migrate to a new OSN, it must be at least as easy to use as any OSNs they currently use and it must be free to use. It remains an open, cross-disciplinary problem to create an OSN that simultaneously: supports user-defined privacy through cryptography, is no more difficult to use than existing OSNs, and can pay for its costs without targetted advertising based on access to private user data.

6.3 The "Best" Solutions

A last question that I consider is whether the solutions I have provided to the problems I have identified in my work are really the best solutions to the problems. In Persona, by design I mandate decentralization to give users some control over their social network account and identity through competition. The decentralized requirement motivates much of my design of Persona. I also choose to use ABE to

facilitate secure OSN communication; though it is likely that secure OSN communication could be provided using an alternate scheme that involves only symmetric and asymmetric cryptography, the simplicity of the ABE abstraction fits the situation of Persona extraordinarily well and provides both users and application designers with intuition about privacy in the system.

As far as I can tell, there is no reason to *not* include Bond Breaker as a component of OSN PKI bootstrapping. Though my study shows that some users have difficulty using Bond Breaker for PKI bootstrapping, it provides an extra option for key exchange that, as my results show, remains difficult to attack. Including Bond Breaker as a component of PKI bootstrapping does not preclude the use of other, potentially better solutions, and merely gives the user another intuitive way to verify identity.

Pseudonymous rendezvous, or at least something that resembles it, appears to be the only solution to providing privacy for many applications. This statement is supported by the fact that many existing systems implement a pseudonymous rendezvous component in an ad hoc way. The real question is whether extracting pseudonymous rendezvous as a modular component has utility or whether these ad hoc solutions are sufficient. My argument that the Twain abstraction is useful is an anecdotal one, the result of consecutive attempts to solve different privacy problems. In each case I returned to the same problem of pseudonymous rendezvous, confident that that was the best solution I could provide. It is my opinion that when the opportunity for reuse presents itself so often and so strongly that abstraction and generalization are clearly the path to pursue to reduce the amount of work that

needs to be done in the future.

6.4 Conclusion

In this work, I have demonstrated technical solutions to many problems that involve manipulating private data on public networks. My solutions are practical and solve a wide range of privacy problems on the Internet today. Collectively my solutions combine to form a social network in which users, not providers, control the exposure of their own private data.

I demonstrated with Persona that private data can be protected even when stored with untrusted third parties, and that many OSN applications only require references to data rather than data contents. I solved the problem of key management, even in difficult settings such as with friends-of-friends, where a group's membership may be unknown.

With Bond Breaker I showed that, even though it is easy for attackers to impersonate users on an OSN to subvert their friends' privacy policies, I can use user-driven shared knowledge to detect and eliminate these attacks. I also demonstrated that I could bootstrap a robust social PKI by propagating public key attestations through a web-of-trust.

Finally, through Twain I was able to broadly expand upon the set of applications to which I can provide users with (hopefully sufficient) privacy policies. I showed that the Twain pseudonymous rendezvous abstraction is a useful way to think about both privacy and connectivity through a variety of examples, some old

and some new. Together with Persona, Twain presents the case that it is generally possible to provide user-defined privacy without sacrificing application functionality.

Appendix A

LoKI Trace Methodology

Our trace collection used two Motorola Droids less than a foot apart from each other in a backpack pocket. The Droids periodically scanned every thirty seconds for wifi access points, discoverable bluetooth devices, and GPS coordinates. The scans were not synchronized; we merely matched up measurements based on the nearest time as an approximation. The traces consisted of two trips walking through the University of Maryland campus, two trips from College Park, MD to Bethesda, MD by way of the Washington, DC metro rail system, and two trips from College Park, MD to Baltimore, MD by car.

Appendix B

CBG Modifications

In order to verify that geolocation information can be derived from network coordinates, we deployed Pyxida on two sets of Planetlab nodes: a low-density set consisting of 30 nodes located throughout the US, and a high-density set consisting of 27 nodes located primarily near the East Coast of the US. We computed two-dimensional network coordinates with a height component, and queried the nodes after convergence. In addition, we measured actual node-to-node latencies to provide a grounds for comparison to the results of CBG based on network coordinate estimated latencies.

Applying the base CBG algorithm to the latency estimations derived from network coordinates was not effective. Due to the inaccuracies of embedding nodes into a coordinate space, the network coordinate latency estimations would underestimate or overestimate the actual measured latency between the two nodes, possibly beyond what should be physically possible. This affects CBG during the landmark RTT vs. distance bestline calculation, where landmark (RTT, distance) pairs would exist below the "baseline", defined as the speed of information along fiber optic cable (1 ms RTT per 100 km). In addition, there were a number of cases where one or more distance contraints were incorrectly derived due to the inaccuracy of the estimated latencies. In the base CBG algorithm, this would return that the geolocation

of a given node could not determined.

To better apply CBG to network coordinates, several modifications were implemented on top of the CBG algorithm as described by Gueye et al. [30]. Any estimated (RTT, Distance) point that lies below the baseline is dismissed as it is a result of underestimation of the latency and cannot physically occur. In addition, outliers in the set of distance constraints are removed based on the area formed as a result of the intersection of the largest number of distance constraints. These modifications relax the CBG algorithm so that it can be applied to network coordinates which may not follow the baseline assumption of CBG.

Bibliography

[1] Alessandro Acquisti and Ralph Gross. Imagined communities: Awareness, information sharing, and privacy on the facebook. In *PET*, 2006.

[2] William Adjie-Winoto, Elliot Schwartz, Hari Balakrishnan, and Jeremy Lilley. The design and implementation of an intentional naming system. *SIGOPS Operating Systems Review*, 2000.

[3] Sharad Agarwal and Jacob R. Lorch. Matchmaking for online games and other latency-sensitive p2p systems. In *SIGCOMM*, 2009.

[4] Shane Ahern, Dean Eckles, Nathaniel S. Good, Simon King, Mor Naaman, and Rahul Nair. Over-exposed?: privacy patterns and considerations in online and mobile photo sharing. In *Human Factors in Computing Systems*, 2007.

[5] Chris Alexander and Ian Goldberg. Improved user authentication in off-the-record messaging. In *WPES*, 2007.

[6] Randy Baden, Adam Bender, Neil Spring, Bobby Bhattacharjee, and Daniel Starin. Persona: An online social network with user-defined privacy. In *SIGCOMM*, 2009.

[7] Randy Baden, Neil Spring, and Bobby Bhattacharjee. Identifying close friends on the internet. In *HotNets*, 2009.

[8] Roberto Baldoni, Leonardo Querzoni, and Antonino Virgillito. Distributed event routing in publish/subscribe communication systems: A survey. Technical report, Dipartimento di Informatica e Sistemistica, UniversitÃă di Roma, 2005.

[9] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *IEEE Symposium on Security and Privacy*, 2007.

[10] Samrat Bhattacharjee, Mostafa H. Ammar, Ellen W. Zegura, Viren Shah, and Zongming Fei. Application-layer anycasting. *IEEE Infocom*, 1997.

[11] Leyla Bilge, Thorsten Strufe, Davide Balzarotti, and Engin Kirda. All your contacts are belong to us: Automated identity theft attacks on social networks. In *WWW*, 2009.

[12] Piero A. Bonatti and Pierangela Samarati. A uniform framework for regulating service access and information release on the web. *Journal of Computer Security*, 2002.

[13] Antonio Carzaniga and Alexander Wolf. Content-based networking: A new communication infrastructure. In *Developing an Infrastructure for Mobile and Wireless Systems*, Lecture Notes in Computer Science. 2002.

[14] Melissa Chase. Multi-authority attribute based encryption. In *TCC*, 2007.

[15] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 1981.

[16] S. Cheshire and M. Krochmal. Multicast DNS. IETF Draft; Expires June 2012, December 2011.

[17] David D. Clark. The design philosophy of the darpa internet protocols. In *SIGCOMM*, 1988.

[18] Landon P. Cox, Angela Dalton, and Varun Marupadi. Smokescreen: flexible privacy controls for presence-sharing. In *MobiSys*, 2007.

[19] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris. Vivaldi: a decentralized network coordinate system. In *SIGCOMM*, 2004.

[20] John Dilley, Bruce Maggs, Jay Parikh, Harald Prokop, Ramesh Sitaraman, and Bill Weihl. Globally distributed content delivery. 2002.

[21] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *USENIX Security*, 2004.

[22] Cynthia Dwork. Differential privacy. In *International Colloquium on Automata, Languages and Programming*, 2006.

[23] Carl M. Ellison. Establishing identity without certification authorities. In *SSYM*, 1996.

[24] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys*, 2003.

[25] Adrienne Felt. Defacing facebook: A security case study. White paper, UC Berkeley, 2007.

[26] David F. Ferraiolo and D. Richard Kuhn. Role-based access controls. In *National Computer Security Conference*, 1992.

[27] Minas Gjoka, Michael Sirivanos, Athina Markopoulou, and Xiaowei Yang. Poking facebook: Characterization of OSN applications. In *WOSN*, 2008.

[28] Ian Avrum Goldberg. *A Pseudonymous Communications Infrastructure for the Internet*. PhD thesis, University of California Berkeley, 2000.

[29] Ralph Gross and Alessandro Acquisti. Information revelation and privacy in online social networks. In *WPES*, 2005.

[30] Bamba Gueye, Artur Ziviani, Mark Crovella, and Serge Fdida. Constraint-based geolocation of internet hosts. *Transactions on Networking*, 2004.

[31] Saikat Guha, Kevin Tang, and Paul Francis. Noyb: Privacy in online social networks. In *WOSN*, 2008.

[32] Krishna P. Gummadi, Stefan Saroiu, and Steven D. Gribble. King: estimating latency between arbitrary internet end hosts. *SIGCOMM Computer Communications Review*, 2002.

[33] Andreas Haeberlen, Petr Kouznetsov, and Peter Druschel. PeerReview: practical accountability for distributed systems. In *SOSP*, 2007.

[34] Jianming He, Wesley W. Chu, and Zhenyu (Victor) Liu. Inferring privacy information from social networks. In *ISI*, 2006.

[35] Pan Hui and Sonja Buchegger. Groupthink and peer pressure: Social influence in online social network groups. In *ASONAM*, 2009.

[36] David P. Jablon. Strong password-only authenticated key exchange. *SIGCOMM Computer Communications Review*, 1996.

[37] Rosie Jones, Ravi Kumar, Bo Pang, and Andrew Tomkins. I know what you did last summer: Query logs and user privacy. In *CIKM*, 2007.

[38] Sewook Jung, Uichin Lee, Alexander Chang, Dae-Ki Cho, and Mario Gerla. Bluetorrent: Cooperative content sharing for bluetooth users. *Pervasive and Mobile Computing*, 2007.

[39] Mohamed Ali Kaafar, Laurent Mathy, Chadi Barakat, Kave Salamatian, Thierry Turletti, and Walid Dabbous. Certified internet coordinates. In *ICCCN*, 2009.

[40] Dina Katabi and John Wroclawski. A framework for scalable global ip-anycast (gia). In *SIGCOMM*, 2000.

[41] Michael L Katz and Carl Shapiro. Network externalities, competition, and compatibility. *American Economic Review*, 75:424–40, June 1985.

[42] Lea Kissner and Dawn Song. Privacy preserving set operations. In *CRYPTO*, 2005.

[43] John Kleinberg. Challenges in social network data: Processes, privacy and paradoxes. In *KDD*, 2007.

[44] Aleksandra Korolova, Rajeev Motwani, Shubha U. Nabar, and Ying Xu. Link privacy in social networks. In *CIKM*, 2008.

[45] Balachandar Krishnamurthy. A measure of online social networks. In *COMSNETS*, 2009.

[46] Balachander Krishnamurthy and Craig Wills. On the leakage of personally identifiable information via online social networks. In *WOSN*, 2009.

[47] Balachander Krishnamurthy and Craig E. Wills. Characterizing privacy in online social networks. In *WOSN*, 2008.

[48] Ieng-Fat Lam, Kuan-Ta Chen, and Ling-Jyh Chen. Involuntary information leakage in social network services. In *IWSEC*, 2008.

[49] Martin Landers, Han Zhang, and Kian-Lee Tan. Peerstore: Better performance by relaxing in peer-to-peer backup. In *P2P*, 2004.

[50] Youngseok Lee. Measured TCP performance in CDMA 1x EV-DO network. In *PAM*, 2006.

[51] Huang Lin, Zhenfu Cao, Xiaohui Liang, and Jun Shao. Secure threshold multi authority attribute based encryption without a central authority. In *INDOCRYPT*, 2008.

[52] Matthew Lucas and Nikita Borisov. flyByNight: Mitigating the privacy risks of social networking. In *WPES*, 2008.

[53] Ben Lynn. *On the implementation of pairing-based cryptosystems*. PhD thesis, Stanford, 2008.

[54] Justin Manweiler, Ryan Scudellari, Zachary Cancio, and Landon P. Cox. We saw each other on the subway: secure, anonymous proximity-based missed connections. In *HotMobile*, 2009.

[55] Justin Manweiler, Ryan Scudellari, and Landon P. Cox. Smile: Encounter-based trust for mobile social services. In *ACM CCS*, 2009.

[56] Huina Mao, Xin Shuai, and Apu Kapadia. Loose tweets: An analysis of privacy leaks on twitter. In *WPES*, 2011.

[57] Suhas Mathur, Robert Miller, Alexander Varshavsky, Wade Trappe, and Narayan Mandayam. Amigo: Proximity-based authentication of mobile devices. In *UbiComp*, 2007.

[58] Suhas Mathur, Robert Miller, Alexander Varshavsky, Wade Trappe, and Narayan Mandayam. ProxiMate: Proximity-based secure pairing using ambient wireless signals. In *MobiSys*, 2011.

[59] Alan Mislove, Meeyoung Cha, Hema Swetha Koppula, Krishna P. Gummadi, Peter Druschel, and Bobby Bhattacharjee. Growth of the Flickr social network. In *WOSN*, 2008.

[60] Alan Mislove, Massimiliano Marcon, Krishna P. Gummadi, Peter Druschel, and Bobby Bhattacharjee. Measurement and analysis of online social networks. In *IMC*, 2007.

[61] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. `http://www.bitcoin.org/bitcoin.pdf`.

[62] Dalit Naor, Moni Naor, and Jeffrey B. Lotspiech. Revocation and tracing schemes for stateless receivers. In *CRYPTO*, 2001.

[63] Shivaramakrishnan Narayan, Parampalli Udaya, and Peter Lee. Identity based signcryption without random oracles. In *International Conference on Security and Cryptography*, 2008.

[64] Arvind Narayanan and Vitaly Shmatikov. Robust de-anonymization of large sparse datasets. In *IEEE Symposium on Security and Privacy*, 2008.

[65] Arvind Narayanan and Vitaly Shmatikov. De-anonymizing social networks. In *IEEE Symposium on Security and Privacy*, 2009.

[66] Arvind Narayanan, Narendran Thiagarajan, Michael Hamburg, Mugdha Lakhani, and Dan Boneh. Location privacy via private proximity testing. In *NDSS*, 2011.

[67] Bryan Parno, Dan Wendlandt, Elaine Shi, Adrian Perrig, Bruce Maggs, and Yih-Chun Hu. Portcullis: Protecting connection setup from denial-of-capability attacks. In *SIGCOMM*, 2007.

[68] C. Partridge, T. Mendez, and W. Milliken. Host Anycasting Service. RFC 1546, 1993.

[69] Anna-Kaisa Pietiläinen, Earl Oliver, Jason LeBrun, George Varghese, and Christophe Diot. Mobiclique: middleware for mobile social networking. In *WOSN*, 2009.

[70] Matthew Pirretti, Patrick Traynor, Patrick McDaniel, and Brent Waters. Secure attribute-based systems. In *ACM CCS*, 2006.

[71] Ariel Rabkin. Personal knowledge questions for fallback authentication: security questions in the era of facebook. In *SOUPS*, 2008.

[72] Anirudh Ramachandran and Nick Feamster. Authenticated out-of-band communication over social links. In *WOSN*, 2008.

[73] Sean Rhea, Brighten Godfrey, Brad Karp, John Kubiatowicz, Sylvia Ratnasamy, Scott Shenker, Ion Stoica, and Harlan Yu. Opendht: a public dht service and its uses. *SIGCOMM Computer Communications Review*, 2005.

[74] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing. Session Traversal Utilities for NAT (STUN). RFC 5389, 2008.

[75] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *Eurocrypt*, 2005.

[76] Sandeep Sarat, Vasileios Pappas, and Andreas Terzis. On the use of anycast in dns. In *SIGMETRICS*, 2005.

[77] Stuart Schechter, A. J. Bernheim Brush, and Serge Egelman. It's no secret: Measuring the security and reliability of authentication via 'secret' questions. In *IEEE Symposium on Security and Privacy*, 2009.

[78] Adi Shamir. How to share a secret. *Communications of the ACM*, 1979.

[79] Umesh Shankar, Kunal Talwar, Jeffrey S. Foster, , and David Wagner. Detecting format-string vulnerabilities with type qualifiers. In *USENIX Security*, 2001.

[80] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *IEEE Symposium on Security and Privacy*, 2000.

[81] William Stallings. The pgp web of trust. *BYTE*, February 1995.

[82] Ryan Stedman, Kayo Yoshida, and Ian Goldberg. A user study of off-the-record messaging. In *SOUPS*, 2008.

[83] Ion Stoica, Daniel Adkins, Shelley Zhuang, Scott Shenker, and Sonesh Surana. Internet indirection infrastructure. *Transactions on Networking*, 2004.

[84] Latanya Sweeney. *Computational Disclosure Control: Theory and Practice*. PhD thesis, Massachusetts Institute of Technology, 2001.

[85] Michael Toomim, Xianhang Zhang, James Fogarty, and James A. Landay. Access control by testing for shared knowledge. In *CHI*, 2008.

[86] Amin Tootoonchian, Kiran K. Gollu, Stefan Saroiu, Yashar Ganjali, and Alec Wolman. Lockr: Social access control for web 2.0. In *WOSN*, 2008.

[87] Nguyen Tran, Jinyang Li, and Lakshminarayanan Subramanian. Collusion-resilient credit-based reputations for peer-to-peer content distribution. In *NetEcon*, 2010.

[88] Patrick Traynor, Kevin Butler, William Enck, and Patrick McDaniel. Realizing massive-scale conditional access systems through attribute-based cryptosystems. In *NDSS*, 2008.

[89] Luis von Ahn, Manuel Blum, Nicholas J. Hopper, and John Langford. Captcha: Using hard ai problems for security. In *Eurocrypt*, 2003.

[90] Long Vu, Indranil Gupta, Jin Liang, and Klara Nahrstedt. Measurement of a large-scale overlay for multimedia streaming. In *HPDC*, 2007.

[91] Steven Euijong Whang and Hector Garcia-Molina. Managing information leakage. Technical report, Stanford University, 2010.

[92] Chung Kei Wong, Mohamed Gouda, and Simon S. Lam. Secure group communications using key graphs. *Transactions on Networking*, 2000.

[93] Wanhong Xu, Xi Zhou, and Lei Li. Inferring privacy information via social relations. In *ICDEW*, 2008.

[94] Andrew C. Yao. Protocols for secure computations. In *Symposium on Foundations of Computer Science*, 1982.

[95] Heng Yin, Dawn Song, Manuel Egele, Engin Kirda, and Christopher Kruegel. Panorama: Capturing system-wide information flow for malware detection and analysis. In *ACM CCS*, 2007.

[96] Ting Yu, Marianne Winslett, and Kent E. Seamons. Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. *Transactions on Information and System Security*, 2003.

[97] Ge Zhong and Urs Hengartner. Toward a distributed k-anonymity protocol for location privacy. In *WPES*, 2008.

[98] ABC News. `http://www.abcnews.go.com/Technology/Story?id=7960020`.

[99] Advanced crypto software collection. `http://acsc.cs.utexas.edu/`.

[100] Android market: 10 billion apps served so far, and another 1 billion each month. `http://techcrunch.com/2011/12/06/android-market-10-billion-apps-served-so-far-and-another-1-billion-each-month/`.

[101] Apple iPhone SDK. `http://developer.apple.com/iphone/`.

[102] Specification of the bluetooth system. `https://www.bluetooth.org/Technical/Specifications/adopted.htm`.

[103] Bump. `http://bu.mp/`.

[104] Re-examining Dropbox and its alternatives. `http://windowssecrets.com/top-story/re-examining-dropbox-and-its-alternatives/`.

[105] Facebook terms of service. `https://www.facebook.com/legal/terms`.

[106] Following data leak, facebook proposes encryption for uids. `http://mashable.com/2010/10/21/facebook-uid-encryption/`.

[107] Free vs. paid Android apps. `http://www.appbrain.com/stats/free-and-paid-android-applications`.

[108] Google terms of service. http://www.google.com/intl/en/policies/terms/.

[109] HTC android flaw leaks smartphone user data. http://www.informationweek.com/news/security/mobile/231700100.

[110] Exactly what does your apple iphone or ipad record about you? http://www.pcworld.com/article/225754/exactly_what_does_your_apple_iphone_or_ipad_record_about_you.html.

[111] Keyslinger. http://www.cylab.cmu.edu/keyslinger/.

[112] Linkedin. http://www.linkedin.com/.

[113] Loopt. http://www.loopt.com/.

[114] PC World. http://www.pcworld.com/article/168462/.

[115] Sony makes it official: Playstation network hacked. http://www.pcworld.com/article/226128/sony_makes_it_official_playstation_network_hacked.html.

[116] Verisign hacked: Security repeatedly breached at key internet operator. http://www.huffingtonpost.com/2012/02/02/verisign-hack_n_1249275.html.

[117] Where I've Been. http://apps.facebook.com/whereivebeen.