Privacy-Enabling Social Networking Over Untrusted Networks

Jonathan Anderson
Computer Laboratory
University of Cambridge
jra40@cl.cam.ac.uk

Claudia Diaz
ESAT/COSIC
K.U. Leuven
claudia.diaz@esat.kuleuven.be

Joseph Bonneau
Computer Laboratory
University of Cambridge
jcb82@cl.cam.ac.uk

Frank Stajano
Computer Laboratory
University of Cambridge
fms27@cl.cam.ac.uk

ABSTRACT

Current social networks require users to place absolute faith in their operators, and the inability of operators to protect users from malicious agents has led to sensitive private information being made public. We propose an architecture for social networking that protects users' social information from both the operator and other network users. This architecture builds a social network out of smart clients and an untrusted central server in a way that removes the need for faith in network operators and gives users control of their privacy.

Categories and Subject Descriptors

C.2.2 [**Computer – Communication Networks**]: Network protocols—*Applications*; C.2.4 [**Computer – Communication Networks**]: Distributed systems—*Client / server, distributed applications, distributed databases*; E.1 [**Data Structures**]: Distributed data structures, Graphs and networks; E.3 [**Data Encryption**]: Public key cryptosystems, Standards; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms

Design, Performance, Security

Keywords

Social networks, privacy, distributed access control

1. INTRODUCTION

Many may not think about it, but using a social network ought to be a sobering activity, as it requires the user to place absolute faith in the Social Network Operator (SNO). First, the social network user provides the SNO with a great deal of personal information which an attacker could use to answer personal knowledge questions and thus impersonate the user [17]. Next, the user tells the SNO who their real-life friends are, which is highly valuable information to those conducting targeted phishing attacks [9]. The user then posts comments and photographs of which some SNOs may claim ownership. Finally, even if it they are legal and appropriate in context, these comments and photographs could cause material harm to the user should they come to light in another context [19] and thus expose the user to the threat of blackmail [16].

SNOs have databases with such information for millions of users, and their incentives—mostly pertaining to growth—may not be aligned with those of their users. Users wishing to protect their personal information must have alternatives which do not rely on the all-knowing SNO: no matter how incompetent or wicked a network operator is, users should be able to expect that their private information is only shared with others when they desire it.

The desire to remove our dependance on the SNO might lead us to consider distributing the central server into a peer-to-peer system. We believe that this approach could have important benefits if we concern ourselves with anonymity properties (invisibility, unlinkability, etc.), but anonymity is well beyond the scope of the current work, as is the challenge of building a reliable, secure and yet high-performance distributed network among untrusting peers.

Instead of building a distributed system, we ask the question: can we define a privacy-enabling architecture for social networking that takes advantage of the simplicity and performance of the centralised client/server model? This architecture should protect personal information from both the SNO and other users, including personal details such as name, gender and contact information, the presence of friendship links and the amount of information that a user has stored in the network. This information must be protected from a number of threats and attackers; we propose a threat model for users' social information and a set of system requirements in Section 2.

Current social networks already attempt to protect this information from other users, but we propose an architecture in Section 3 that would also protect private information from the operator and its partners, ensuring that invasive features [20] and malicious applications [14] do not expose users' private information.

We describe our proof-of-concept implementation of some of the architecture's features in Section 4 and discuss the expected performance of the system, including measurements of some functionality, in Section 5.

2. REQUIREMENTS

The requirements for a privacy-enabling social network are twofold: it must safeguard the privacy of its users while providing the social utility that they have become accustomed to and rightfully expect to increase as the technology matures.

2.1 Privacy Requirements

The users of a privacy-enabling social network will expect the system to protect their personal information from unauthorised access. Personal information includes both *content* (profile information, messages, photos, etc.) and *links* (friendship, group membership, social browsing history, etc.). The system should protect this information from two types of attackers: the network operator and other network users.

The operator of a centralised social network or the underlying network infrastructure is a powerful adversary. In a centralised system, we cannot prevent the SNO from performing traffic analysis or denial of service attacks, but we can address the threats of the operator viewing or modifying user information surreptitiously.

Other users of the social network have more limited abilities than the network operator: they cannot observe all the actions of other users within the network. We can, therefore, act to mitigate their potential threats. The most obvious threat is simply that they might view content which they are not authorised to view, but our system should also prevent them from:

- learning about the existence of content to which they do not have access;
- learning social graph data (e.g. friendship links) of other users and
- modifying or deleting another user's content.

2.2 Functional Requirements

While respecting these privacy requirements, a social network must enable users to interact with each other via social links, direct communication, shared content and other means which we cannot currently envision.

2.2.1 Extensibility

We cannot predict the innovative ways in which users will adapt the network to their needs [5]; our core task should be to create a social API rather than merely a fixed set of features. The proposed client architecture should allow third-party developers to provide applications which enhance the functionality of the network and users to install such applications without yielding control of their personal information.

2.2.2 Social Links

In order for a social network to be social, it must support interaction among friends, family, colleagues and acquaintances. In order to establish connections, a user of a social network must advertise themselves to other users. By default, users of a site should be invisible to all other users, but they should be able to advertise themselves in three ways:

1. By making some personal information (e.g. name, photo) available to the entire Web;
2. By sending social network information via existing communication channels (e.g. IM, e-mail);
3. Via social relationships in the network itself.

The first method is similar to how users currently find each other on social networks, whereas the second method is a way to bootstrap a social network from existing networks, which some of today's social networks do not provide.[1] The third is the most powerful – at the application layer, within a secure communications channel, users could "suggest" relationships to their friends based on e.g. common interests or background. This would provide similar functionality to mechanisms on existing social networks without revealing information to the SNO.

A function which a privacy-enabling social network should provide—and which current social networks do not–is a means for users to bind online identities to those of their real-world friends. Today's social networks are already used to impersonate important politicians for comedic purposes [13], but the potential to use social networks for highly targeted phishing attacks is tremendous. Users should therefore be provided with an easy-to-use yet cryptographically strong means of verifying identity using out-of-band signalling (in person, over the phone, etc.).

2.2.3 Posting Personal Information

Posting personal content in a traditional client-server model is as straightforward as uploading the content to the server and providing other users with access to it. If the server is untrusted, however, we must take the responsibility for access control away from the server and give it to its clients.

Clients are capable of performing distributed access control but, once access has been granted, there is no way to prevent others from copying content, any more than a traditional social network can prevent users from saving photos to their hard disk from their web browser. This is not a problem of technical enforcement, but of social contract. Client software could provide users with a means of watermarking content to identify the source of "leaks", but people can usually copy what they see and repeat what they hear.

2.2.4 Messaging

Users should be able to send secure messages to each other. Depending on social context, delivery could be instantaneous or delay-tolerant, verifiable or repudiable.

2.2.5 Joint Content

One of the most interesting subjects in the debate over online privacy is the case of *joint content*: content that was created and posted by one person, possibly in response to

[1]For instance, it is impossible for two Facebook users who have opted out of searchability to connect with each other.

other content, which involves another person's name, likeness, opinions or comments. For instance, if a user is "tagged" in a Facebook photograph, they have the right to remove the tag—which recognises the stake that they hold in the content—but tagging another user's photo also requires the permission of the person who posted the photo.

In a privacy-enabling social network, there will be no way for the central server to identify and remove photographs that a particular user appears in. This is not a technical shortcoming, but a fundamental social limitation: one cannot ensure that while in public, no photographs are taken of them and later published in a newspaper or circulated via e-mail. What a privacy-enabling social network *can* do, however, is specify a convention on the association of content with identity.

The existence of content in a data repository is meaningless unless there are links to the content. When managing one's online persona, what matters is not the existence of embarrassing photographs, but whether or not a potential employer sees the photographs when searching for "Jonathan Anderson". Our system cannot overcome the fundamental social limitation inherent in the former, but it can stipulate that "well-behaved" clients ought to ignore links and tags about a person that have not been published or otherwise approved by their subject. By way of analogy, one cannot stop others from spreading slanderous rumours by word of mouth, but one can be sure their their web site, publications, etc. do not reference them or otherwise give them credence.

3. ARCHITECTURE

We propose a client-server social networking architecture in which the server is untrusted, providing only availability, and clients are responsible for their own confidentiality and integrity. Similar topologies have been applied to network file systems in the past [10].

In this architecture, the server is both untrusted and simple to operate. Because it is untrusted, there is no need for the client software to be written by the storage provider; multiple independent vendors could produce competing client software, as long as they follow the same specifications. Because it is simple, it would require little computational power or maintenance; it could, in fact, merely be an interface to a content delivery network.

The clients in this architecture will be composed of several software layers. These are, from top to bottom, the application layer, the data structures layer, the cryptographic layer and the network layer.

3.1 Application Layer

Applications will run inside a secure sandbox that will limit their access to user data and communication channels. This is, in computing terms, a very old concept [11], yet it is a dramatic improvement beyond current platforms, whose applications run on third-party servers and can often store and share private information freely [14].

Within its sandbox, an application will be prevented from accessing user information or other applications unless that access is mediated by the user's client. The client will expose an API which will be governed by a security policy, which should be restrictive by default yet unobtrusive, built using techniques such as "security by designation" [22].

Just as a system call API can include a (relatively) small number of fundamental, atomic operations, so should our application API provide basic social functions that user-created applications can combine in new and interesting ways. Designing security APIs is a difficult problem [1] but, if done well, combinations of atomic operations will not affect our reasoning about privacy and security.

3.2 Data Structures Layer

In order to prevent other users from learning how much content a user has posted (and thus discovering how much content is hidden from them), it is essential that user content exist as a collection of discrete blocks, whose linkages are known only to those who are authorised to read them.

These blocks will contain user content and links to other blocks. Links need to be hidden in order to satisfy the requirement that other users must not be able to see how much information they do not have access to. Such hiding is accomplished by cryptographic means which are described in Section 3.3.

As each block is decrypted and its links discerned, a tree of blocks and their links can be compiled. Walking this tree provides the client with all content that it is able to access about a particular user; the root node of the tree can thus be viewed as the interface to all of the user's content[2].

3.3 Cryptography Layer

The cryptographic layer of the client software provides both confidentiality, protecting content from those not authorised to view it, and integrity, verifying that other users' content is genuine. Specific tasks that it must perform include managing keys, verifying identity, encrypting private information and signing content.

3.3.1 Identity Verification

Our access control system assumes shared public keys between pairs of users who are sharing content. There are several ways in which cryptography can be used to support binding online identities to real-world ones.

One rigorous but tedious approach is for users to exchange the entire hex string of their public keys in a face-to-face meeting. The identity verification itself is thus performed in the traditional, social way, but we do not expect exchanging large base-64 numbers to be a popular activity.

A more realistic approach could involve the distribution of cryptoIDs [15], which are public key fingerprints of a manageable size[3]. After the exchange, users would enter each others' CryptoIDs into their client software, which would look up, retrieve and verify their full public keys. Again, real-world identities are verified in the traditional way, but a CryptoID fingerprint is small enough to exchange via business card or even the back of a napkin.

If users initially connect via an untrusted channel, e.g. e-mail, website, IM or the network itself, verification can be done through a simple multi-channel protocol. Assuming that we have at our disposal a low-capacity bidirectional authentic channel (a face-to-face meeting, talking over the telephone, etc.), we can use the multi-channel authentication

[2]Multi-faceted identity is also possible: trees can be constructed with different branches (or even different roots) presented to different peers.

[3]For instance, the CryptoID `f3v4g.ifcen.r3rj5.embx8` is "a little longer than most email addresses, a little shorter than most postal addresses, and about the same size as a credit card number plus its 4-digit expiration date." [15]

Algorithm 1 Mutual Authentication Protocol

```
$A \longrightarrow B$ : $K_{\mathrm{A}}$
$B \longrightarrow A$ : $K_{\mathrm{B}}$
$A \longrightarrow B$ : $H\left(A \mid K_{\mathrm{A}} \mid K_{\mathrm{B}} \mid R_a \mid K_a\right)$
$B \longrightarrow A$ : $H\left(B \mid K_{\mathrm{A}} \mid K_{\mathrm{B}} \mid R_b \mid K_b\right)$
$A \overset{\text{offline}}{\longrightarrow} B$ : $R_a$
$B \overset{\text{offline}}{\longrightarrow} A$ : $R_b$
$A \longrightarrow B$ : $K_a$
$B \longrightarrow A$ : $K_b$
$A \overset{\text{offline}}{\longrightarrow} B$ : outcome
$B \overset{\text{offline}}{\longrightarrow} A$ : outcome
```

protocol of Wong and Stajano [21] shown in Algorithm 1.

In this protocol, A and B are the identities of Alice and Bob (which could be a cryptoID, UUID, e-mail address, etc.) and K_A and K_B are their public keys. R_a and R_b are short random nonces that Alice and Bob generate, while K_a and K_b are long random nonces.

The security of this system does not rely on the length of the short nonces, or even on their secrecy; rather, it depends on *data origin authenticity*—Bob knows that Alice really was the person who provided him with R_a, and vice versa. Even if an attacker Eve can replace messages 3 and 4 with hashes of chosen public keys and long nonces, she cannot choose R_a or R_b and is unlikely to guess them. This is true even if these nonces are very short—a 4-digit PIN would be perfectly acceptable to prevent this attack.

Finally, users and their client software might opt in some situations to depend on "traditional" online identity verification mechanisms such third-party certification or a Web of Trust. Such mechanisms do not offer the same level of assurance as meeting with a friend face to face, but they may be appropriate for some social or business relationships.

3.3.2 Link Hiding

Link hiding can be accomplished very simply via stream cipher encryption. Every block of user information should end with padding that is indistinguishable from random:

```
03 00 4d 0b 59 7a e5 b0 7a bf 89 c8 f6 b0 2d
74 76 2d 30 64 67 9a 42 f6 34 15 bc 66 71 91
2a 34 0e e6 45 c4 ff 8f d7 90 95 4a e3 a8 2e
```

Different portions of this padding may, however, be constructed from different stream cipher keystreams. Thus, when one user attempts to decrypt it, she will see the result:

```
1e fb L  I  N  K  :  a  b  c  .. .. 92 71 44
99 1c ff bf d9 5a e1 03 08 8e 7d 9b c2 45 56
aa dd 0e 64 fc 7f a3 c4 77 77 e6 a0 81 c4 5a
```

Whereas another user, following the same procedure but using a different key, will see:

```
ad e6 e1 69 fd 4e 70 3c da ce f8 c6 94 0f e7
3c 6b 66 c5 39 6c 1c 74 c1 14 ef 53 L  I  N
K  :  d  e  f  .. .. 70 32 22 12 37 9d 92 e4
```

These links can be hidden in random locations within the padding so that clients cannot determine how many links exist within the padding and are hidden from them.

3.3.3 Key Management

The most straightforward system of key management would be to simply encrypt all content multiple times, once per authorised user. Encrypting every block of information to the public key or shared key of every recipient, however, would be a very heavy computational burden.

A more practical scheme is to encrypt blocks under a symmetric key which can be easily changed every time the block's access control list changes. This symmetric key could then be made available to other users via a hierarchical scheme involving shared group keys. This is a space-time trade-off: more blocks of information must be created, but the cost of these extra blocks (a small amount of extra storage and network transfer) will sometimes be less than that of performing many cryptographic operations (higher operation latency).

Like the hidden links described in Section 3.3.2, the encryption of K_{group} should be done in such a way that one user of the group cannot determine how many other users there are, using random locations within a header whose content is indistinguishable from random.

3.3.4 Joint Content

There are several different schemes to enable users to jointly post content, depending on the underlying security model desired. A publicy-postable "wall"' feature can be implemented by linking to a block which stores the contents of the wall and providing some others with the capability to update that block. Tags or comments that require approval will take the form of a message which includes the joint content itself and a hash of its context (the original content, previous comments, etc.), signed by the creators of both the original and the joint content. This gives the owner of the original content the ability to endorse joint content and the creator of the joint content the ability to specify the context that their content should not be taken out of.

3.3.5 Messaging

Delay-tolerant messaging *à la* e-mail or Facebook messaging is relatively straightforward: the sender's client need only create a portion of their profile which only the recipient can see, and embed the message within it. The only question that arises is, should the message be digitally signed or not? Digital signatures are useful for providing assurance to the recipient that a message is genuine, but in some situations, anonymity and plausible deniability are more important. Helping users make this decision is an important usability question for the future.

For instant messaging functionality, we could set up a peer-to-peer IM session between clients protected by a well-known protocol such as Off-The-Record messaging [2, 18].

3.4 Network Layer

The interactions between client and server are very simple: clients store blocks of content on the server, which makes them available for other clients to access. The access control on the server is very limited: it need only prevent overwrites. The API between client and server, therefore, could consist of just two commands:

- $x = \mathbf{GET}(ID)$
- $C = \mathbf{POST}(ID, x, C)$

The **GET** command retrieves a block from the server. No read access control is performed at the server: all blocks are publicly readable, but they will usually be encrypted.

The **POST** command lets users post a block of content x having the unique identifier ID. The response to this command is a capability C which allows the block to be overwritten in the future. Without holding such a capability, a client can only post to an as-yet unused UUID.

These simple functions are all that is required of the network layer, but the addition of other commands could aid performance without compromising privacy. For example, delay-tolerant messaging via profile blocks could be made more efficient if the server provided a limited signalling channel to each user which signified "user X has sent you a message; you may wish to check for updates in their profile".

4. IMPLEMENTATION

We have implemented some[4] of the functionality of this system as a Java application which, without requiring local installation, behaves like a familiar web interface. This application, which runs via Java Web Start, acts as a Web server, serving HTML, JavaScript and Ajax content to the user's browser.

The Java Virtual Machine (JVM) provides us with a nearly-ubiquitous platform on which to write client software, whether using the application-as-web-server model, a traditional desktop environment or a JavaME on a mobile platform.

Java's security policy architecture also allows the JVM to run different pieces of code with different levels of permission [7]. Applications are thus run as plugin components with no permission to perform system operations such as open network sockets or draw graphics on the screen[5]. All such interactions must go through our application API, where privacy and security policies can be enforced.

5. PERFORMANCE

The proposed architecture should scale more easily than existing social networks, since the central server's storage requirement is no larger, but the computational demands placed on it are far smaller.

Clients will perform more computation than they do in current networks, but this work only scales as a factor of the number of users a particular client interacts with, not the number of clients in the entire system.

To consider the speed of the system, let us assume that a user, Alice, has d friends whose profiles she wishes to view, and each of those profile has n blocks of equal size arranged in a tree with a branching factor of b. We will assume that Alice has the right to see all dn blocks, but that they are encrypted so that she must decrypt a block before she can retrieve its children.

[4]We have prepared a proof of concept implementation of the technologies which we believe required demonstration: a custom classloader running in Java Web Start which can load plugins with limited permissions, a local Web server which communicates with browser-side Ajax and cryptographic code for performance testing.

[5]Unfortunately, Java's security policy does not allow us to restrict access to the current system time; if it did, we could even shut down covert channels of information flow among malicious applications [11]. Still, changing the flow of private information from unrestricted to covert-channel-only is a very significant improvement.

We define t_{b} to be the time required to download a block from the server, t_{pk} to be the time required to perform public-key decryption and t_{sk} to be the time required to perform symmetric-key encryption or decryption. We assume that Alice can perform cryptographic operations in parallel with network operations, as long as the required information has all been downloaded. We also assume that Alice has no cache of any of her friends' information blocks, so she must download them all, starting with the d blocks which comprise the roots of the profile trees.

As Alice downloads the roots of her friends' profiles, she must decrypt them. The first decryption will be a public-key operation to retrieve symmetric keys which have been granted to Alice. These operations require time t_{pk} , and the first such operation will begin once the first block has been downloaded at time t_{b}.

At time $t_{\mathrm{b}} + t_{\mathrm{pk}}$, then, Alice will have downloaded and decrypted one friend's root node, which will point her towards another b blocks. Sets of b blocks will continue to be added to the download queue as each root block is decrypted, and all d root blocks be downloaded and decrypted at time t_{d}:

$$t_{\mathrm{d}} = \begin{cases} dt_{\mathrm{b}} + t_{\mathrm{pk}} & t_{\mathrm{pk}} \leq t_{\mathrm{b}} \\ t_{\mathrm{b}} + dt_{\mathrm{pk}} & t_{\mathrm{pk}} > t_{\mathrm{b}} \end{cases}$$

Using standard Java cryptography libraries (SunJCE v1.7) on a desktop PC (Intel Core2 Quad CPU Q6600 @ 2.4 GHz), we were able to perform 2048b RSA encryption at 200 kB/s and 1024b encryption at over 600 kB/s. Decryption progressed at ≈30 kB/s with a 1024b key and 7 kB/s with a 2048b key, so it is likely that $t_{\mathrm{pk}} \geq t_{\mathrm{b}}$.

As long as $t_{\mathrm{pk}} < dt_{\mathrm{b}}$, however, the network I/O queue will not empty, as the first root block which is decrypted will yield URLs for b non-root blocks to download, and each of these blocks will yield b more. Almost all of these blocks will be encrypted with a symmetric key, and symmetric key performance is much better than public-key: using the above computer installation, we were able to encrypt over 35 MB of data per second using 128b AES, which is almost instantaneous when compared to typical download rates on consumer equipment.

Thus, as long as $t_{\mathrm{pk}} < dt_{\mathrm{b}}$, the task of retrieving the profiles of a user's friends will be I/O-bound, even in the worst case where no symmetric keys have been cached. If some keys are cached, even this requirement of $t_{\mathrm{pk}} < dt_{\mathrm{b}}$ will be relaxed.

6. RELATED WORK

Distributed social networks have been proposed by Buchegger and Datta with a view to develop distributed access control in the future [3, 4].

Guha et al have proposed a privacy scheme for social networks in which Facebook users host the profile information of others, and a particular user's profile can be reconstituted from their friends via a keyed function [8]. Lucas and Borisov proposed simply encrypting user information [12], but neither of these schemes addressed the problem of protecting the links between users.

Felt and Evans suggested *privacy-by-proxy*, which limits applications' initial access to personal information [6], though it doesn't enforce policies about application behaviour once the application has a user's personal information.

The SUNDR network file system created by Li et al [10] is an example of a client-server architecture in which clients use cryptography to share a general-purpose filesystem on an untrusted server. This system uses cryptography heavily, yet its performance is comparable to established network file systems.

7. CONCLUSION

We have proposed an architecture for privacy-enabling social networking which would satisfy users' requirements for existing functionality, as well as provide a platform on which sandboxed applications could be written to provide unforeseen functionality in a privacy-enabling way. The architecture is based on simple client-server interactions which transfer encrypted blocks, blocks which when decrypted can be used to reconstitute all of the content that another user has shared with the client.

The architecture provides means to protect private information against discovery and disclosure, not just by other users, but by the network operator. Social graph information is also protected against discovery by other users, but it is impossible to prevent a centralised server from performing traffic analysis; further work should include protection against traffic analysis through decentralisation.

Acknowledgements

This work was partially supported by the Rothermere Foundation, the Gates Cambridge Trust, the IWT SBO ADAPID project, the Concerted Research Action (GOA) Ambiorics 2005/11 of the Flemish Government and the IAP Programme P6/26 BCRYPT.

8. REFERENCES

[1] Bond, M. *Understanding Security APIs*. PhD thesis, University of Cambridge, Jan 2004.

[2] Borisov, N., Goldberg, I., and Brewer, E. Off-the-Record Communication, or, Why Not To Use PGP. In *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society* (2004), pp. 77 – 84.

[3] Buchegger, S., and Datta, A. A Case for P2P Infrastructure for Social Networks - Opportunities & Challenges. In *Proceedings of the Sixth International Conference on Wireless On-demand Network Systems and Services* (2009).

[4] Buchegger, S., Schioberg, D., Vu, L.-H., and Datta, A. PeerSoN: P2P Social Networking — Early Experiences and Insights. In *Proceedings of the Second Annual EuroSys Workshop on Social Network Systems* (Mar 2009).

[5] danah michele boyd. *Taken Out of Context – American Teen Sociality in Networked Publics*. PhD thesis, University of California, Berkeley, 2008.

[6] Felt, A., and Evans, D. Privacy Protection for Social Networking Platforms. In *Proceedings of Web 2.0 Security and Privacy 2008* (2008).

[7] Gong, L., Mueller, M., Prafullchandra, H., and Schemers, R. Going Beyond the Sandbox: An Overview of the New Security Architecture in the Java Development Kit 1.2. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems* (1997).

[8] Guha, S., Tang, K., and Francis, P. NOYB: Privacy in Online Social Networks. In *Proceedings of the First Workshop on Online Social Networks* (Aug 2008), pp. 49 – 54.

[9] Jagatic, T., Johnson, N., Jakobsson, M., and Menczer, F. Social Phishing. *Communications of the ACM 50*, 10 (Oct 2007), 94 – 100.

[10] Li, J., Krohn, M., Mazieres, D., and Shasha, D. Secure untrusted data repository (SUNDR). *the 6th Symposium on Operating Systems Design and Implementation* (2004).

[11] Lipner, S. B. A Comment on the Confinement Problem. *ACM SIGOPS Operating Systems Review 9*, 5 (1975), 192 – 196.

[12] Lucas, M., and Borisov, N. FlyByNight: Mitigating the Privacy Risks of Social Networking. *the 7th ACM Workshop on Privacy in the Electronic Society* (Oct 2008).

[13] McGonigle, B. Some profiles on MySpace.com not what they seem. `http://www.boston.com/news/nation/washington/articles/2006/10/16/some_profiles_on_myspacecom_not_what_they_seem/`, October 2006. The Boston Globe.

[14] Mills, E. Facebook suspends app that permitted peephole. `http://news.cnet.com/8301-10784_3-9977762-7.html`, Jun 2008. CNET News.

[15] Perrin, T. Public key distribution through "cryptoIDs". In *Proceedings of the 2003 Workshop on New Security Paradigms* (Aug 2003), pp. 87 – 102.

[16] Pilkington, E. Blackmail claim stirs fears over Facebook. `http://www.guardian.co.uk/business/2007/jul/16/usnews.news`, Jul 2007. The Guardian.

[17] Rabkin, A. Personal knowledge questions for fallback authentication: Security questions in the era of Facebook. In *Proceedings of Symposium on Usable Privacy and Security* (2008), pp. 13 – 23.

[18] Raimondo, M., Gennaro, R., and Krawczyk, H. Secure Off-the-Record Messaging. *the 2005 ACM Workshop on Privacy in the Electronic Society* (Nov 2005), 81 – 89.

[19] Randall, D., and Richards, V. Facebook can ruin your life. And so can MySpace, Bebo... `http://www.independent.co.uk/life-style/gadgets-and-tech/news/facebook-can-ruin-your-life-and-so-can\-myspace-bebo-780521.html`, February 2008. The Independent.

[20] Story, L., and Stone, B. Facebook Retreats on Online Tracking. `http://www.nytimes.com/2007/11/30/technology/30face.html`, Nov 2007. The New York Times.

[21] Wong, F.-L., and Stajano, F. Multi-channel Protocols. In *Proceedings of the International Workshop on Security Protocols* (2005), pp. 112 – 127.

[22] Yee, K.-P. Aligning security and usability. *IEEE Security and Privacy Magazine 2*, 5 (2004), 48 – 55.