Privacy engineering for social networks

Jonathan Anderson

University of Cambridge
Trinity College

July 2012

This dissertation is submitted for the degree of Doctor of Philosophy

Declaration

This dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration except where specifically indicated in the text.

This dissertation does not exceed the regulation length of 60,000 words, including tables and footnotes.

Privacy engineering for social networks

Jonathan Anderson

In this dissertation, I enumerate several privacy problems in online social networks (OSNs) and describe a system called Footlights that addresses them. Footlights is a platform for distributed social applications that allows users to control the sharing of private information. It is designed to compete with the performance of today's centralised OSNs, but it does not trust centralised infrastructure to enforce security properties.

Based on several socio-technical scenarios, I extract concrete technical problems to be solved and show how the existing research literature does not solve them. Addressing these problems fully would fundamentally change users' interactions with OSNs, providing real control over online sharing.

I also demonstrate that today's OSNs do not provide this control: both user data and the social graph are vulnerable to practical privacy attacks.

Footlights' storage substrate provides private, scalable, sharable storage using untrusted servers. Under realistic assumptions, the direct cost of operating this storage system is less than one US dollar per user-year. It is the foundation for a practical shared filesystem, a *perfectly unobservable* communications channel and a distributed application platform.

The Footlights application platform allows third-party developers to write social applications without direct access to users' private data. Applications run in a confined environment with a private-by-default security model: applications can only access user information with explicit user consent. I demonstrate that practical applications can be written on this platform.

The security of Footlights user data is based on public-key cryptography, but users are able to log in to the system without carrying a private key on a hardware token. Instead, users authenticate to a set of *authentication agents* using a weak secret such as a user-chosen password or randomly-assigned 4-digit number. The protocol is designed to be secure even in the face of malicious authentication agents.

Acknowledgements

I gratefully acknowledge the sponsorship of the Rothermere Foundation. The 2nd Viscount Rothermere, first Chancellor of Memorial University of Newfoundland, endowed the Rothermere Fellowship to aid Memorial students in their pursuit of higher degrees in the United Kingdom. Without this generous support, I would not have been able to pursue my PhD in Cambridge.

I am also grateful for the sponsorship I received from the Natural Sciences and Engineering Research Council of Canada (NSERC). The US Defence Advanced Research Projects Agency (DARPA) supported projects that I was involved with during my PhD, although that work is not part of this dissertation.

I thank my supervisor, Frank Stajano, for his guidance and mentorship during my PhD. You gave me freedom to explore my research horizons but kept me grounded in reality when I needed to communicate those ideas or even when I needed to Just Finish. I remember clearly the advice you gave me on the first day I sat in your office; I hope to share it with my own students one day.

They say it takes a village to raise a child. My development as a researcher has been immeasurably spurred by collaborations and conversations with many colleagues. Joseph Bonneau, Claudia Diaz and I collaborated on the work that grew into this dissertation's subject. Robert Watson once poked his head in my office door to ask, "have you ever hacked OS kernels?"; the subsequent collaboration has greatly influenced my thinking about operating systems and application confinement. Luke Church helped open my eyes to alternative viewpoints on security, as well as the existence of users as people rather than rhetorical devices. I am grateful to co-authors Ben Laurie, Kris Kennaway, George Danezis, Sören Preibusch and all of the CTSRD team. My thinking on both security and the nature of research has been aided immeasurably by early advice I received from Ross Anderson, Bruce Christianson and Saar Drimer. I have also received very helpful feedback on drafts of this dissertation from Ross Anderson, Robert Watson, Joseph Bonneau, Anil Madhavapeddy, Richard Clayton, Wei Ming Khoo and Bjoern Zeeb. Thank you all.

I would also like to thank my friends Tom & Julie and Will & Jackie, whose footprints along the doctoral road helped me trace the path to my own PhD.

Thanks to Mom & Dad for buying me Lego and for having patience when I took things apart and couldn't put them back together again. You've always supported my creative and academic pursuits, encouraged me to follow my dreams and believed that I could get there. Thank you.

I also thank Earl & Barbara and Russell & Cara for believing in me and encouraging us to go to Cambridge, even though it took Chrissy so very far away.

Finally, to Chrissy, my wife: thank you for your love, support, encouragement and constant companionship. When I wasn't sure I could finish this work, you were there with words of comfort, strength and purpose. I can't imagine a better friend. I've loved every day we've spent together, and I look forward to loving every day we spend together in the years to come.

Soli Deo gloria.

Author Publications

Journals

[1] R. N. M. Watson, J. Anderson, B. Laurie, and K. Kennaway. A taste of Capsicum: practical capabilities for UNIX. *Communications of the ACM*, 55(3):97, Mar. 2012. doi:10.1145/2093548.2093572.

Conferences

[2] J. Anderson, C. Diaz, J. Bonneau, and F. Stajano. Privacy-enabling social networking over untrusted networks. In *WOSN '09: Proceedings of the Second ACM Workshop on Online Social Networks*. ACM, Aug. 2009. doi:10.1145/1592665.1592667.

[3] J. Anderson and F. Stajano. Not that kind of friend: misleading divergences between online social networks and real-world social protocols. In *SPW '09: Proceedings of the Seventeenth International Workshop on Security Protocols*, Apr. 2009. URL: http://www.cl.cam.ac.uk/~jra40/publications/2009/SPW-misleading-divergences.pdf.

[4] J. Anderson and F. Stajano. On storing private keys "in the cloud". In *SPW 2010: Proceedings of the Eighteenth International Workshop on Security Protocols*, Mar. 2010. URL: http://www.cl.cam.ac.uk/~jra40/publications/2010/SPW-key-storage.pdf.

[5] J. Anderson, F. Stajano, and R. N. M. Watson. How to keep bad papers out of conferences (with minimum reviewer effort). In *SPW 2011: Proceedings of the Nineteenth International Workshop on Security Protocols*, Mar. 2011. doi:10.1007/978-3-642-25867-1_34.

[6] J. Anderson and R. N. M. Watson. Stayin' alive: aliveness as an alternative to authentication. In *SPW 2012: Proceedings of the Twentieth Interna-*

tional Workshop on Security Protocols, Apr. 2012. URL: `http://www.cl.cam.ac.uk/~jra40/publications/2012/SPW-stayin-alive.pdf`.

[7] J. Bonneau, J. Anderson, R. J. Anderson, and F. Stajano. Eight friends are enough: social graph approximation via public listings. In *SNS '09: Proceedings of the Second ACM EuroSys Workshop on Social Network Systems*, pages 13–18. ACM, Mar. 2009. `doi:10.1145/1578002.1578005`.

[8] J. Bonneau, J. Anderson, and G. Danezis. Prying data out of a social network. In *Proceedings of the 2009 International Conference on Advances in Social Networks Analysis and Mining (2009)*, pages 249–254, 2009. `doi:10.1109/ASONAM.2009.45`.

[9] R. N. M. Watson, J. Anderson, B. Laurie, and K. Kennaway. Capsicum: practical capabilities for UNIX. In *Proceedings of the 19th USENIX Security Symposium*. USENIX Association, Aug. 2010. URL: `http://portal.acm.org/citation.cfm?id=1929820.1929824`.

Workshops (no proceedings)

[10] J. Anderson. Psychic routing: upper bounds on routing in private DTNs. In *Hot Topics in Privacy Enhancing Technologies (HotPETS)*, 2011. URL: `http://petsymposium.org/2011/papers/hotpets11-final9Anderson.pdf`.

[11] J. Anderson, J. Bonneau, and F. Stajano. Security APIs for online applications. In *ASA-3: Third International Workshop on Analysing Security APIs*, July 2009. URL: `http://www.cl.cam.ac.uk/~jra40/publications/2009/ASA-security-apis-for-online-applications.pdf`.

[12] J. Anderson, J. Bonneau, and F. Stajano. Inglorious installers: security in the application marketplace. In *WEIS '10: The Ninth Workshop on the Economics of Information Security*, pages 1–47, 2010. URL: `http://www.cl.cam.ac.uk/~jra40/publications/2010/WEIS-inglorious-installers.pdf`.

[13] J. BONNEAU, J. ANDERSON, AND L. CHURCH. Privacy suites: shared privacy for social networks (poster). In *SOUPS '09: Symposium on Usable Privacy and Security*, 2009. URL: `http://cups.cs.cmu.edu/soups/2009/posters/p13-bonneau.pdf`.

[14] L. CHURCH, J. ANDERSON, J. BONNEAU, AND F. STAJANO. Privacy stories: confidence in privacy behaviors through end user programming (poster). In *SOUPS '09: Symposium on Usable Privacy and Security*, 2009. URL: `http://cups.cs.cmu.edu/soups/2009/posters/p3-church.pdf`.

[15] R. N. M. WATSON AND J. ANDERSON. Connecting the dot dots: model checking concurrency in Capsicum. In *International Workshop on Analysing Security APIs*, July 2010. URL: `http://www.cl.cam.ac.uk/~jra40/publications/2010/ASA-capsicum-dot-dots.pdf`.

[16] R. N. M. WATSON, P. G. NEUMAN, J. WOODRUFF, J. ANDERSON, R. J. ANDERSON, N. DAVE, B. LAURIE, S. W. MOORE, S. J. MURDOCH, P. PAEPS, M. ROE, AND H. SAIDI. CHERI: a research platform deconflating hardware virtualization and protection. In *RESoLVE'12: Runtime Environments, Systems, Layering and Virtualized Environments*, Mar. 2012. URL: `http://www.dcs.gla.ac.uk/conferences/resolve12/papers/session1_paper3.pdf`.

MAGAZINE ARTICLES

[17] R. N. M. WATSON, J. ANDERSON, B. LAURIE, AND K. KENNAWAY. Introducing Capsicum: practical capabilities for UNIX. *;login:—The USENIX Magazine*, 35(6):7–17, Dec. 2010. URL: `https://www.usenix.org/system/files/login/articles/watson.pdf`.

WHITE PAPERS, BLOG POSTS AND PRESS RELEASES

[18] J. BONNEAU AND J. ANDERSON. Think of the children [online]. Dec. 2008. URL: `http://www.lightbluetouchpaper.org/2008/12/12/think-of-the-children/`.

[19] J. Bonneau, J. Anderson, R. J. Anderson, and R. Clayton. Democracy theatre on Facebook [online]. Mar. 2009. URL: http://www.lightbluetouchpaper.org/2009/03/29/commentary-on-facebooks-terms-of-service/.

[20] J. Bonneau, J. Anderson, A. Lewis, and F. Stajano. Attack of the zombie photos [online]. May 2009. URL: http://www.lightbluetouchpaper.org/2009/05/20/attack-of-the-zombie-photos/.

[21] J. Bonneau, J. Anderson, F. Stajano, and R. J. Anderson. Facebook consultation as much of a sham as their democracy. Light Blue Touchpaper, Apr. 2009. URL: http://www.cl.cam.ac.uk/~jra40/publications/2009/facebook-press-release.pdf.

[22] J. Bonneau, S. Preibusch, J. Anderson, R. Clayton, and R. J. Anderson. Comments on Facebook's proposed governance scheme [online]. Mar. 2009. URL: http://www.cl.cam.ac.uk/~jra40/publications/2009/LBT-facebook-governance.pdf.

[23] J. Bonneau, S. Preibusch, J. Anderson, R. Clayton, and R. J. Anderson. The curtain opens on Facebook's democracy theatre [online]. Apr. 2009. URL: http://www.lightbluetouchpaper.org/2009/04/17/the-curtain-opens-on-facebooks-democracy-theatre/.

Contents

Author Publications ix

1 Introduction 1

1.1 A socio-technical problem . . . 3

1.1.1 Jack . . . 3

1.1.2 CBSA . . . 4

1.1.3 Facebook . . . 4

1.1.4 Application developers . . . 5

1.1.5 Sophie . . . 5

1.2 Contributions . . . 5

1.3 Publications . . . 8

2 Background 13

2.1 Definitions . . . 13

2.2 Problems . . . 16

2.2.1 Explicit expressions of user intent . . . 16

2.2.2 Multi-faceted identity . . . 17

2.2.3 High availability . . . 18

2.2.4 Untrusted infrastructure . . . 18

2.2.5 Social applications . . . 19

2.2.6 Cost . . . 20

2.2.7 Linkability and anonymity . . . 21

2.3 Related work . . . 22

2.3.1 Encryption within OSNs . . . 22

2.3.2 Distributed social networks . . . 24

2.3.3 Solutions out of context . . . 27

2.3.4 Privacy policy specification . . . 28

3 Antisocial networks **31**
3.1 User data privacy . . . 31
3.1.1 Forced open: evolving defaults in privacy settings . . . 32
3.1.1.1 Use of default settings . . . 32
3.1.1.2 A history of changing defaults . . . 33
3.1.2 Failed open: advertisers and private information . . . 41
3.1.3 Left open: application access to private user data . . . 42
3.2 Social graph privacy . . . 46
3.2.1 Public search listings . . . 47
3.2.2 Public listings model . . . 48
3.2.3 Graph data . . . 49
3.2.4 Approximations of graph characteristics . . . 51
3.2.4.1 Reachability and shortest paths . . . 51
3.2.4.2 Centrality and influence . . . 53
3.2.5 Related work . . . 56
3.2.6 Summary . . . 56
3.3 Summary of contributions . . . 57

4 Sharable storage **59**
4.1 Route of trust to root of trust . . . 62
4.2 Content-Addressed Store . . . 65
4.3 Immutable blocks . . . 66
4.3.1 Naming . . . 67
4.3.2 Structure . . . 69
4.3.3 Encryption . . . 70
4.4 Mutable names . . . 72
4.5 Filesystem . . . 76
4.5.1 A root by any other name . . . 76
4.5.2 Cross-filesystem sharing . . . 77
4.5.3 Writing to shared filesystems . . . 78
4.5.4 Garbage collection . . . 78
4.6 Cost . . . 79
4.7 Pre-cacheing and traffic analysis . . . 82

4.8 Covert communication 85
4.8.1 Link hiding 87
4.8.2 Message forwarding 87
4.9 Related work 91
4.10 Summary of contributions 97

5 Distributed Social Applications 99
5.1 Model of computation 100
5.1.1 Securing user data 101
5.1.2 Social DRM 103
5.1.3 Footlights 105
5.2 Front end 107
5.2.1 Unprivileged application context 108
5.2.1.1 Visual container 108
5.2.1.2 Intra-application communication 110
5.2.1.3 Static content 111
5.2.2 Privileged supervisor 112
5.3 Back end 116
5.3.1 Applications 116
5.3.1.1 Distribution 116
5.3.1.2 Initialisation 118
5.3.1.3 File services 118
5.3.1.4 Ajax service 119
5.3.2 Confinement 120
5.3.3 Static content 121
5.3.4 Kernel API 123
5.3.4.1 Files and directories 123
5.3.4.2 Powerboxes 128
5.3.4.3 Synchronous communication 129
5.3.5 Persistent key-value store 130
5.4 Example applications 131
5.4.1 Basic demo 131
5.4.2 Malicious demo 133

5.4.3 Tic-tac-toe . . . 134
5.4.4 Photos . . . 135
5.4.5 File manager . . . 137
5.5 Related work . . . 138
5.6 Summary of contributions . . . 140

6 Distributed authentication 141
6.1 Problems . . . 142
6.1.1 The assertion problem . . . 142
6.1.2 Weak secrets . . . 143
6.1.3 Location independence . . . 143
6.1.4 Limited trust . . . 144
6.1.5 Plausible deniability . . . 144
6.2 Principals . . . 144
6.2.1 Insiders . . . 145
6.2.2 Outsiders . . . 145
6.3 Protocols . . . 146
6.3.1 Trusted third party . . . 147
6.3.2 Semi-trusted storage . . . 148
6.3.3 Secret sharing . . . 149
6.3.4 Collision-rich password hashing . . . 150
6.3.4.1 Large-M attack . . . 151
6.3.4.2 Impostor identity disclosure attack . . . 151
6.3.5 Collision-rich identity hashing . . . 152
6.4 Passwords and probabilities . . . 152
6.4.1 Uniform distribution . . . 154
6.4.2 Non-uniform distribution . . . 155
6.4.3 Outsider dictionary attack . . . 158
6.4.3.1 Uniform password distribution . . . 158
6.4.3.2 Non-uniform password distribution . . . 161
6.4.4 Insider dictionary attack . . . 163
6.4.4.1 Uniform password distribution . . . 163
6.4.4.2 Non-uniform password distribution . . . 164

6.4.5 Summary 164
6.5 Revocation 167
6.6 Related work 168
6.7 Summary of contributions 172

7 Conclusions **175**
7.1 Problems 175
7.2 Footlights 176
7.3 Future work 179
7.4 Summary 183

A Content Delivery Networks **185**

Bibliography **189**

Other References **219**

List of Figures

1.1 High-level overview of the Footlights social platform. 7

2.1 Direct and indirect access to user data. 20

3.1 Facebook privacy timeline 35
3.2 A sampled graph G_k with $k = 1$. 49
3.3 Route lengths and reachability in university networks. 51
3.4 Route length approximation under k-sampling (Columbia). 52
3.5 Betweenness centrality in four university networks. 54
3.6 Attacker influence over messages in the Harvard network. 55

4.1 Footlights and its static storage layers. 60
4.2 A block with three links and 3,210 B of user data. 68
4.3 Binary representation of a CAS link. 69
4.4 Resolving a canonical name. 74
4.5 Bytes required to store photos in the Footlights filesystem. 80
4.6 Cost of operating the Footlights storage service. 80
4.7 Cost of basing Footlights on Amazon Web Services. 81
4.8 Embedding covert content in random padding. 85
4.9 Block upload/download is equivalent to multicast messaging. . . 88
4.10 Random padding viewed as spare message-carrying capacity. . . 88

5.1 Today's OSN as an application platform. 103
5.2 Footlights applications run locally. 105
5.3 Applications are split into front-end and back-end components. . 107
5.4 An application's front-end context. 108
5.5 Creating a UI element with the DOM and Footlights APIs. 109
5.6 Communication over a multiplexed HTTP channel. 111
5.7 Two ways of displaying the user's name. 115
5.8 A Footlights powerbox. 115

5.9 Sharing a directory with another user. 129
5.10 A basic demo application. 132
5.11 A Tic-Tac-Toe demo application. 134
5.12 A basic photo manager. 135
5.13 A basic file manager. 137
5.14 Sharing a directory with the File Manager application. 137

6.1 Password domain, distribution and equivalence classes. 153
6.2 Possible presentation of a random word from a small dictionary. . 155
6.3 Frequency of RockYou passwords v. a power law distribution. . . 156
6.4 Probability distribution of h_M values for RockYou passwords. . . 157
6.5 Probability of successfully attacking a uniform distribution. . . . 160
6.6 Probability of successfully attacking the RockYou distribution. . . 162
6.7 The RockYou distribution, sampled for several values of M. . . . 165
6.8 Attacking the sampled RockYou distribution. 166

List of Tables

3.1 Basic statistics from early Facebook networks. 50
3.2 Increase in shortest-path route lengths caused by *k*-sampling. . . 53
3.3 Node compromises required to influence socially-routed messages. 56

A.1 Pricing of Amazon S3 and CloudFront services. 186
A.2 Pricing of Google App Engine storage and content delivery. . . . 186
A.3 Pricing of Rackspace storage and content delivery. 187
A.4 Pricing of NetDNA content delivery. 187

Abbreviations

ACL access control list; a specification of privileges granted to principals

API application programming interface

ASCII American standard code for information interchange; a scheme for encoding characters required by the English language

CAS content-addressed store; a storage system that names files according to their contents (see URN)

CDN content delivery network; a global network used to deliver data to geographically disparate locations with low latency

DES data encryption standard; an historic block cipher

DHT distributed hash table; a CAS spread across a P2P network

DOM document object model; an API for interacting with web pages

DRM digital rights management; a set of techniques for preventing users from copying content (e.g. games, music, videos)

DSL digital subscriber line; a technology for home Internet service

DTN delay tolerant network; a network that does not require end-to-end connectivity between hosts in order to forward messages

FBML Facebook markup language; formerly used by Facebook applications to describe web content and user information

FQL Facebook query language; a SQL-like language used by applications to retrieve information about users from Facebook

HTTP hypertext transfer protocol; used for communicating web content

HTTPS	HTTP over SSL; a security extension to HTTP
IP	internet protocol; the protocol used to communicate among computers on the internet *or* the address used for this communication
IPC	inter-process communication; an explicit protocol for communication among programs that are separated from each other and therefore cannot use implicit protocols (e.g. function calls)
JAR	Java archive; a format for bundling Java programs and other application content into a single file
JSON	JavaScript object notation; a widely-used scheme for encoding data
JVM	Java virtual machine; software that runs programs written in the Java programming language
MAC	mandatory access control; an enforcement mechanism that allows system administrators to specify system security policies
MLS	multi-level security; a MAC policy that prevents confidential information from flowing to unauthorised principals
NFS	network file system; a file system shared between users on a network and hosted on a centralised server
OS	operating system; low-level system software that mediates application access to underlying resources such as physical hardware
OSN	online social network
P2P	peer-to-peer; a network topology that has no central control point
PDF	Portable Document Format; a file format for document interchange
PET	privacy enhancing technology
PGP	pretty good privacy; a protocol for asymmetric-key encryption and trust management

PIN personal identification number; a short, numeric password

RSA an asymmetric cryptography scheme developed by Rivest, Shamir and Adleman [209]

SQL structured query language; a language for interacting with databases

SSL secure sockets layer; a technique used for securing a variety of internet protocols, including HTTP; now named TLS

TCB trusted computing base; the portions of a system that users and applications must trust; see the definition of "Trust" on page 15

TLS transport layer security; the modern replacement for SSL

UI user interface; a channel through which a user and system communicate, e.g. visual "windows" manipulated with a keyboard and mouse

UID user identifier; a string that uniquely identifies a user

URI uniform resource identifer; a standardised format for naming network-accessible objects

URL uniform resource locator; a URI that describes an object's location (e.g. a file name on a particular Web server)

URN uniform resource name; a URI that names an object by an intrinsic property of the object (e.g. its content)

UX user experience; a holistic view of user interaction with a system

1 Introduction

I wanted to create an environment where people could share whatever information they wanted, but also have control over whom they shared that information with.

Mark Zuckerberg, September 2006 [310]

Today, hundreds of millions of people communicate via Online Social Networks (OSNs) such as Facebook, Google+ and Renren. These services allow users to express personal attributes and relationships, as well as communicate with other users in front of a virtual audience. Users can manage events, sending invitations to them, hosting discussion before them and providing a repository for photographs during and after them. They can share photographs and tag them with locations and identities, often assisted by OSN-provided facial recognition. They can "check in" to locations, declaring their presence and that of others with them; this facilitates spontaneous offline meetings and enriches users' daily routines. Social applications enrich the experience by integrating others' activities: "your friends have read this article, would you like to read it too?"

Hundreds of millions have flocked to these services, sharing private information with the services and their friends. In some cases, however, their information has been shared more widely than they intended. Private information has been shared with third-party applications and with advertisers, sometimes against the expressed intentions of users. On Facebook, advertisers can use "Sponsored Stories" to remind my friends again and again that I once expressed positive sentiments about a chain of taco restaurants. Those same "friends", who include my family, co-workers and possibly my future employers, can see things I've done and places I've been, sometimes without my involvement or permission: by default, a single authorisation gives friends the power to tell each other where I am. Unless I take steps to prevent it, all of my "friends" can read things that I once said within a private context, organised in a near-public timeline that stretches back to the day of my birth. Applications used by my "friends" can access information that I shared with Facebook on the understanding that it would be visible to no-one. The developers of those applications can harvest my personal information and sell it to third-party data aggregators. As Chapter 3 will show, some already have; there is no technical protection to stop this behaviour, nor is there a technical remedy once it is done.

This dissertation considers these problems and more, then demonstrates that the status quo is not the only way of doing online social networking. While it is not a trivial task — as Section 2.3, "Related work" will show — it is possible to build online social networks that do not require users to give up their privacy, place absolute faith in service providers or suffer from peer-to-peer performance penalties. Users can *rely on* centralised infrastructure to provide the availability they expect without *trusting* it to enforce their intentions for sharing.

The thesis of this dissertation is that practical online social networking is possible without mandatory privacy or performance penalties: users can choose their own privacy–performance trade-offs. I demonstrate this by describing a hybrid centralised–distributed OSN called Footlights that improves on the state of the art in user data protection. Footlights allows users to choose their own privacy and performance trade-offs *and* enjoy the benefits of social applications. Its prototype implementation is open-source, available under the permissive Apache License [24] from `https://github.com/trombonehero/Footlights`.

This technical contribution allows users to adopt a new kind of OSN with a new business model, one in which one US dollar per user-year of direct costs must be recouped via privacy-preserving advertising, direct payment or some other method. However, I do not claim to have "solved" the complex socio-technical issue of privacy in social networking. Instead, I provide a technical system that allows users to more effectively direct the flow of their private information according to their own social and economic choices.

1.1 A SOCIO-TECHNICAL PROBLEM

> *There's no technical solution for gossip.*
>
> Matthew Ringel, LiveJournal user, 2003 [352]

Many approaches can be taken to the study of privacy in online social networks. This dissertation will not explore legal or ethical dimensions of the problem, nor will it propose technical "quick fixes" to social problems. As I have argued elsewhere, the attempt to impose simplistic technical models on complex social systems has caused some of the current mismatches between the behaviour of OSNs and users' expectations [3]. This dissertation begins by considering mismatches between users' expressions of sharing intent and the actual behaviour of online social networks. These mismatches are illustrated with several real-life scenarios. In these scenarios, technology alone would not solve the problem, but there is a technical part to be played in the socio-technical system.

1.1.1 JACK

An early example of divergence between user expectation and reality comes from the blogging site LiveJournal. In 2003, the website SecurityFocus reported that security issues with the site had led to private information becoming public [352]. In one particular example, a user referred to as "Jack" experienced an account compromise, in which an attacker re-posted private LiveJournal entries publicly, revealing sensitive information about Jack's friends, including a discussion with one friend about her relationship with her employer. Jack's account may have been compromised by a password-sniffing attacker — at the time, LiveJournal authentication was unencrypted — but one of Jack's friends may have simply guessed Jack's password. Technical measures can be used to prevent the former scenario: it has become an industry standard for websites to provide their users with encrypted connections, as announced by Twitter [338] and Google's search [324] and GMail [356] teams, even if it is not always enabled by default as on Facebook [299]. The latter compromise scenario is more complicated, however: technology can provide additional authentication factors such as hardware tokens or personal knowledge questions, but their use is subject to non-technical considerations such as cost and shared experience.

1.1.2 CBSA

There is less causal ambiguity in the 2007 case of Canada Border Services Agency (CBSA) recruits whose photographs leaked from Facebook, as reported by the Canadian Broadcasting Corporation [269, 268]. In that case, photos and comments were posted on Facebook which were described by the Minister of Public Safety as "simply not acceptable". There is no evidence of technical failure on the part of Facebook: it seems that the photos were simply posted with permissive access controls. In this case, better technology may not have changed users' decisions to make the comments or post the photos, but it may have been able to better capture users' intent as to how widely that content should be shared. However, the decision to deploy such technology instead of permissive default settings is not a purely technical decision; it is grounded in the economics of the OSN business.

1.1.3 FACEBOOK

The opposite problem can be observed in the next scenario. In 2009, Facebook disregarded the explicitly-expressed privacy preferences of its users and decided to make all users' friend lists public [302]. Since "the overwhelming majority of people who use Facebook already make most or all of this information available to everyone" — the default setting — the change only affected those users who had taken explicit action to express restrictive privacy preferences and limit the sharing of their friend lists. The subsequent user backlash led to Facebook providing a new visibility control that allowed friend lists to be hidden from other users' view, but the information is "still publicly available [...] and can be accessed by applications" [296]. This case features a simple technical change: a handful of privacy controls were disabled. The controversy came about because Facebook made the change against the express wishes of those affected by it: those whose information is now shared publicly.

1.1.4 APPLICATION DEVELOPERS

In 2010, the Wall Street Journal found that several developers of popular social applications were transmitting user information to third-party advertisers and commercial data aggregators [282, 275]. In this case, applications contravened Facebook's terms of service, but the contravention was not rectified until the Wall Street Journal made the application behaviour public. This leakage of user information beyond the intent of the OSN came about because of the trust structure of today's OSNs: third-party application developers must be trusted with private data in order for applications — as they are currently structured — to perform useful tasks.

1.1.5 SOPHIE

The final scenario is that of "Sophie", a person with whom I have corresponded personally. Sophie is technically apt, a facile computer user with a PhD in science, but she does not trust centralised online services with her personal data. As a result, she cannot take advantage of the services or social applications offered by centralised OSNs, nor can she converse easily with those who use OSNs as their primary mode of communication.

1.2 CONTRIBUTIONS

The scenarios in the previous section provide windows into a complex sociotechnical system. In this dissertation, I do not attempt to "solve" social problems by pressing them into the mold of graph theory or other easily-expressed mathematical and computer science concepts. Rather, in Chapter 2, I extract from the above scenarios a set of technical problems whose solutions would allow users to change the balance of the overall complex system that is online privacy. In this section, I outline my solution to these problems.

In this dissertation, I demonstrate some of the privacy problems with today's online social networks and show how social networking could be done differently, providing solutions to the problems enumerated in Section 2.2 and potentially allowing users to change the outcomes they would experience in the real-life scenarios of Section 1.1. I show in Section 2.3 that existing proposals in the research literature do not address all of these problems.

In Chapter 3, I show that there is a cost to today's nominally free services. Privacy is sacrificed, both for individual users (Section 3.1) and the social graph as a whole (Section 3.2). OSNs dictate terms and conditions under which they can change their users' implicit privacy contract, pushing users into ever-less-private social interaction (§3.1.1). Services have a history of leaking users' private information both to advertisers (§3.1.2) and to application developers (§3.1.3). OSN operators unilaterally decide that some personal information must be visible to the world, driving growth but sacrificing privacy for all participants in the social graph (Section 3.2).

In order to address the problems in Section 2.2, I present Footlights, a privacy-enabling platform for untrusted social applications that does not trust centralised infrastructure. Not only are applications and intrastructure untrusted, even core parts of the system — both software and protocols — are open for inspection and verification. Users do not even need to trust Footlights itself, as will be discussed in the definition of "Trust" on page 15.

The name "Footlights" comes from Goffman's theatrical analogy for social interaction [115], itself based on Shakespeare's declaration that "all the world's a stage, and men and women merely players" [215]. In a theatre, footlights define the conventional physical boundary between the stage and the audience; in my work, the Footlights system helps users to create performances or definitions of self which are displayed to an audience chosen by the performer. Some users are invited to Goffman's metaphorical backstage, to see behind the scenes of differing or contradictory performances, but those seated in the audience should not be able to see any evidence that the performer is hanging from wires or that the backdrop is made of cheap black stuff. This analogy is the source of the name Footlights, but it should also be seen as *homage* to the Cambridge Footlights, a real-world theatrical ensemble that has launched some of the world's leading acting careers.

Footlights uses an architecture that is radically different from today's OSNs, yet still practical. It provides the features afforded by today's OSNs and competes with their performance but does not impose their privacy costs. A high-

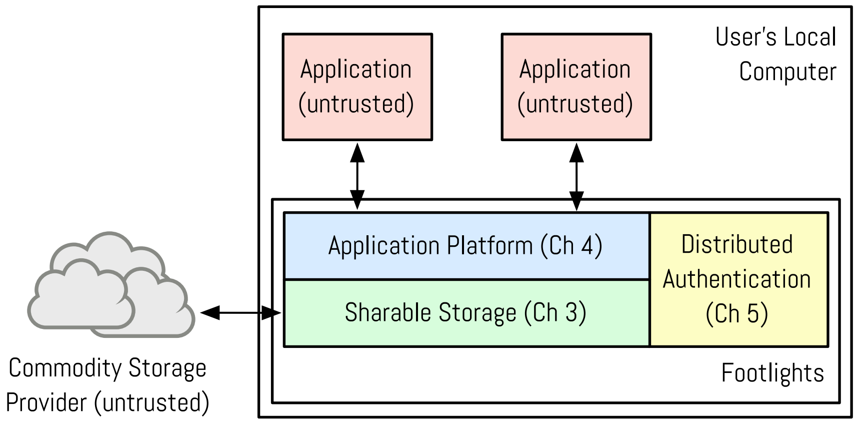


Figure 1.1: High-level overview of the Footlights social platform.

level diagram of Footlights is shown in Figure 1.1.

The Footlights storage substrate provides private, scalable, sharable storage using untrusted servers (Chapter 4). Under realistic assumptions, the direct cost of operating this storage system is less than one US dollar per user-year. It is the foundation for a practical shared filesystem, a *perfectly unobservable* communications channel and a distributed application platform.

The Footlights application platform (Chapter 5) allows third-party developers to write social applications without direct access to users' private information. Distributed applications run under a very different model of computation from today's OSNs: applications execute in a confined environment under the user's control. The application security model is private by default: applications can only access user information with user consent. This consent is inferred from natural user actions, rather than "cancel or allow" prompting. Within this constrained environment, the platform provides traditional Operating System services to applications, allowing them to perform useful computation; I demonstrate how practical applications can be written for this platform.

The security of Footlights user data is based on cryptography, so no centralised service must be trusted to enforce users' security policies. Unlike existing approaches to cryptographic authentication, however, Footlights does not require users to remember strong passwords or carry private keys on hardware tokens. Instead, I leverage the distributed social system: users authenticate to a set of *authentication agents* using a weak secret such as a user-chosen password or a randomly-assigned 4-digit number (Chapter 6). Even in the face of malicious authentication agents guessing these weak secrets, users can choose parameters for the authentication scheme to provide a chosen level of security.

1.3 PUBLICATIONS

In the course of my PhD research, I have co-authored nine peer-reviewed papers, one in *Communications of the ACM*. I have also co-authored seven contributions to workshops and poster sessions without proceedings, one magazine article and six contributions to the Security Group blog, Light Blue Touchpaper.

I started my PhD in 2008 examining "virtual worlds", of which online social networks are a subset. Early on, I began examining the security of OSNs and was joined by Joseph Bonneau in work that led to several publications. In 2008, we detailed numerous privacy failings in a social network designed for children in a posting to the Security Group blog [18]. In 2009, we used data from Facebook in two publications: "Eight friends are enough", a study of the information leaked by Facebook's Public Listings feature published at the Second ACM EuroSys Workshop on Social Networks Systems (SNS) [7] and "Prying Data out of a Social Network", a summary of various methods that we used to extract information from the service, published at the 2009 International Conference on Advances in Social Networks Analysis and Mining (ASONAM) [8]. I have based much of Chapter 3 on the work that I did for these papers: the collaborative work that we undertook in extracting data from the social network, the individual work that I did exploring the Facebook application API and the individual analysis that I performed approximating two characteristics of the *publicly-sampled* social graph (Section 3.2).

During this time I also co-authored several Light Blue Touchpaper articles on the subject of Facebook's proposed new governance models with Joseph Bonneau, Sören Preibusch, Ross Anderson and Frank Stajano [19, 21, 22, 23]. These were collaborative works, in one case driven by a request from the Open Rights Group to provide comments on their behalf, and Joseph Bonneau was the lead author. I also assisted Joseph, along with Andrew Lewis and Frank Stajano, in testing the revocation properties of various OSNs' content delivery networks, which was discussed in a 2009 post on Light Blue Touchpaper [20].

Based on all of this study, I published a paper at the 2009 International Workshop on Security Protocols (SPW), describing some of the misleading divergences between online and real-world social networks and proposing design principles to mitigate the problems in existing OSNs [3]. I developed these ideas further into a software architecture and partial prototype that was published later in 2009 at the second ACM Workshop on Social Networks (WOSN) [2]. This work was the product of many collegial debates with co-authors Claudia Diaz, Joseph Bonneau and Frank Stajano. This architecture would eventually develop into Footlights, a system that I have been building ever since.

As the Footlights architecture developed, I explored ideas around its *perfectly unobservable* communications channels at the 2011 Hot Topics in Privacy Enhancing Technologies (HotPETs) workshop [10]. This work has influenced my thinking about the covert channels described in Chapter 4, but its material has not been directly included in this dissertation.

I developed the ideas in Chapter 6 after a conversation with Sonja Buchegger of the PeerSoN project [65] in which we discussed PeerSoN's need for a form of authentication that was reliant neither on centralised infrastructure nor users' ability to memorise or carry long private keys. This work was published at the 2010 International Workshop on Security Protocols (SPW) [4]; Chapter 6 is an expansion of these ideas with a more in-depth evaluation.

I discussed the Footlights security API in a presentation, co-authored with Joseph Bonneau, given at the third annual Analysing Security APIs workshop (ASA) [11]. I explored the security and privacy differences between OSNs and traditional application marketplaces, assisted by Joseph and Frank Stajano, at the 2010 Workshop on the Economics of Information Security (WEIS) [12]. Neither of these papers have been directly included in the dissertation, but they have influenced Footlights design decisions detailed in Chapter 5.

I have also collaborated with Luke Church and Joseph Bonneau to produce two posters for the 2009 Symposium on Usable Privacy and Security. One of these posters, of which Luke was the primary author and I was the secondary author, focused on applying user programming techniques to the problem of privacy policy specification and included a pilot user study [14]. The other, of which I was the third author, described the idea of "privacy suites", expert-supplied default privacy settings that could be applied by OSN users [13]. Neither of these posters' material is included in this dissertation.

I also published other papers during my PhD that have little to do with online social networks directly but which have influenced my thinking about security issues generally.

In 2010, I joined Robert Watson's Capsicum project, exploring how the capability model of security could be applied to traditional UNIX operating systems. This work was published at the 2010 USENIX Security Symposium, where it won a Best Student Paper award [9]. We also summarised the work for USENIX *;login:* magazine [17] and explored a particular security API problem that we encountered at the fourth Analysing Security APIs workshop [15]. This work was collaborative in nature, but Robert Watson was the initiator, chief architect and primary programmer for the project. We have since merged Capsicum into mainline FreeBSD, and Capsicum has been featured as a research highlight in the March 2010 issue of *Communications of the ACM* [1]. The constraints imposed on Capsicum by current hardware architectures led to the four-year CTSRD project to re-consider aspects of hardware design that affect security. So far, the CTSRD project has resulted in one workshop paper describing a new CPU architecture called CHERI (Capability Hardware Enhanced RISC Instructions); this was presented at the 2012 Runtime Environments, Systems, Layering and Virtualized Environments (RESoLVE 2012) workshop [16]. Although it is not directly included in this dissertation, my work on Capsicum and CTSRD have significantly influenced my thinking on application confinement, which is a major focus of Chapter 5.

I have also published two additional papers at the International Workshop on Security Protocols (SPW). The first, co-authored in 2011 with Frank Stajano and Robert Watson, considers how automated analysis of citation graphs might help members of Programme Committees to better focus their time when reviewing submissions to conferences and journals [5]. The second, co-authored with Robert Watson and presented at the 2012 workshop, describes modifications to the Kerberos protocol that could allow users to eschew traditional authentication mechanisms in certain situations, instead presenting evidence to authentication servers that the user is “a person who is currently alive” [6].

2

Background

This dissertation assumes a general computer science background for the reader. In addition to this background, readers will require a functional familiarity with cryptographic hash functions, symmetric-key encryption and public-key encryption and signatures. Readers unfamiliar with these cryptographic primitives may wish to consult Katz and Lindell's *Introduction to Modern Cryptography* [140] or Menezes, van Oorschot and Vanstone's *Handbook of Applied Cryptography* [174].

Beyond these cryptographic primitives, the reader will find in this chapter some definitions of terms (Section 2.1) used throughout the dissertation and an enumeration of specific technical problems (Section 2.2) to be addressed by any system attempting to supplant today's online social networks (OSNs). As stated in Section 1.1, addressing these problems will not "solve" privacy, but solutions would provide a technical framework in which users can change the parameters of the socio-technical system that is privacy in online social networks.

Finally, I provide an overview of some of the related work in this area (Section 2.3) and show why it does not solve the problems in Section 2.2.

2.1 Definitions

This section defines several terms which appear frequently in the discourse around privacy and online social networks. Many of these terms will figure in this dissertation, but I begin with a term that purposefully does not.

Social media A term that confuses meaning by conflating several scarcely-related concepts while excluding very-related others. Common use of the phrase "social media" encompasses Twitter and Facebook, which have radically different usage models, but not blogs and e-mail, which are both media and social in nature. Instead of this phrase, I use the more specific terms defined below.

Personal broadcast medium A communications medium which affords every user the ability to broadcast to a large, potentially global audience. This includes online social networks and microblogs as well as more traditional media such as blogs, mailing lists, Internet Relay Chat (IRC) and Usenet. Such media often provide a way to initiate a two-way conversation with readers, but their primary usage is broadcast and "everyone has a microphone" [349].

Online Social Network (OSN) An online experience that centres around the sharing of digital artefacts such as public messages ("wall" posts), photographs and group events. Users can discuss and collaboratively edit the experience according to an access control scheme that is based on stable identities. This definition excludes most blogs and microblogs, but includes those blogs that allow users to comment on articles based on stable identities and permit the limiting of viewership.

Facebook The current OSN market leader. This dissertation will describe issues around OSNs abstractly, but many examples will be drawn from Facebook. Steve Jobs overstated the case only slightly when he said, "we talk about social networks in the plural, but I don't see anybody other than Facebook out there. Just Facebook, they are dominating this" [272]. Facebook is the dominant player in the market, so the way that it treats user data is more significant than the ways that other networks do, and the practices that have fueled its explosive growth deserve particular scrutiny. Where Facebook's practices are unique or of particular interest, I highlight them as such, but wherever possible, I describe the behaviour of Facebook and its competitors abstractly, using the term OSN.

Privacy Privacy is a complex social good with no single agreed definition. From Warren and Brandeis' "right to be let alone" [228] to the German constitutional court's "right to informational self-determination" [133], there are varying interpretations of what privacy means and how it interacts with other rights. When does my right to informational self-determination override others' right to free speech? These legal questions have not been uniformly settled across jurisdiction and are well outside my domain of expertise. Instead of considering them further, therefore, this dissertation focuses on aspects of the problem where clear technical contributions can be made: keeping a user-driven distinction between private and public information, as defined below.

Public Accessible by any person or computer without special authorisation. This includes information that is restricted to users of a particular system if no authorisation is required to become a user. It also includes information that is technically available but labelled with a restrictive information-flow label, e.g. covered by a `robots.txt` file [323] or available under contractual restrictions.

Private Information that a user intends to be non-public. This definition does not require users to explicitly label information as private. The question of intent also steers clear of legal definitions of Personally Identifiable Information (PII) or Personal Data, most of which were written in a time before Big Data and the ability to identify users from an "anonymised" social graph as shown by Backstrom et al. [35], Narayanan et al. [188, 189] and Wondracek et al. [233].

User It is sometimes said in coffee breaks at academic security workshops that "social networks are for people who don't care about privacy" or that "they get what they ask for". This has not been the finding of research by Acquisti, boyd, Gross, Krishnamurthy and others into actual use of privacy settings over time [26, 59, 125, 147], nor has it been my experience in speaking with OSN users. For the purposes of this dissertation, I assume that the User is a person who uses an OSN's information-sharing capabilities and who may choose to create Private information (as defined above).

Trust I adopt the definition of trust recorded by Christianson and Harbison: to say that A trusts B is to say that "B has the ability to violate A's security policy", or more intuitively, “B has the power to get A sacked” [74][1]. In particular, trust is about power and choice rather than truth: a user may choose or be forced to trust an untrustworthy system or a system may be designed around untrusted parties who are in fact trustworthy. Trust is also separable from *reliance*: a user may rely on a server to perform a service, but that need not imply trust if the user can independently verify that the service was performed correctly. This definition of trust is quite different from that implied by the famous Russian proverb, “trust but verify” [320].

Honest but curious A model of behaviour for a protocol participant who is relied on but not trusted. This model was first introduced under a different name in 1987 by Goldreich, Micali and Wigderson [116]. They described a “passive adversary” who would perform a protocol correctly but “try to compute more than their due share of knowledge”. In 1995, Beimel and Chor coined the label above: they modeled participants in a secret-sharing scheme as honest (“they follow their protocols”) but curious (“after the protocol has ended some of them can collude and try to gain some partial information on the secret”) [38].

2.2 PROBLEMS

Based on these definitions, I define several technical problems to be solved by a privacy-enabling online social network. Solving these problems will not *guarantee* a different outcome for the subjects of all the scenarios in Section 1.1, but it will *allow* the users in those scenarios to realise more desirable outcomes.

2.2.1 EXPLICIT EXPRESSIONS OF USER INTENT

In today’s OSNs, user actions can have *hidden dependencies*, using the Cognitive Dimensions nomenclature described by Blackwell, Green et al. [50, 122]: clicking a button may both express a sentiment (e.g. “I like this”) and confer invisible privilege to the author of the application containing the button. The

[1]Christianson credits Robert Morris with introducing the security policy definition at the first Security Protocols Workshop in 1993 [73]. The “getting-sacked” variant, in inverted form, is credited by Ross Anderson to another comment made by Roger Needham at the same workshop [29]. Unfortunately, the transcripts of this workshop have been lost.

fact that the user clicked the button may be visible to many more people than the user would have chosen to reveal the fact to. The application author may be permitted to alter the user's profile and advertise to their "friends". By contrast, a privacy-enabling OSN should seek clear and explicit expressions of user intent. Service providers, advertising platforms and third-party application developers should only see a user's private information if the user expresses a decision to share it.

These expressions of intent may be — and in Footlights, are — translated into cryptographic operations, but users should not need a grounding in cryptography or computer security to use the system. Wherever possible, users' sharing intent should be inferred *transparently*, using existing user actions rather than new prompts or dialogues.

A solution to this problem would change the outcome of scenario "Facebook" on page 4, in which users' private information was made public, whether automatically through permissive defaults or accidentally through a user interface that failed to capture user intent.

2.2.2 MULTI-FACETED IDENTITY

In the case of the CBSA recruits whose private photos leaked to the public (scenario "CBSA" on page 4), a different outcome may have been experienced if the recruits were better able to segregate their personal and professional online personas. Today's OSN users may segregate different facets of their identities by using different OSNs for different roles [346], but difficulties arise when there is overlap between friends, family, co-workers, supervisor and supervisees.

Instead of a coarse partitioning between OSNs, a privacy-enabling OSN should support a fine-grained expression of user identity: users should be able to express different facets of an online persona within different contexts. Users must be able to assert these facets of identity — e.g. "I post this comment in response to that photo" — without leveraging trusted authentication infrastructure or requiring users to carry hardware cryptographic tokens. Furthermore, a user's various "audiences" should not be able to infer the existence of other audiences or content, as observed by Greschbach et al. [123].

Multi-faceted identity must not impose a burden of explicit key management on users: cryptography must be "under the hood", implemented as a mapping from higher-level concepts such as "users" and "accounts".

2.2.3 High availability

When users share content with other users, it must be reliably stored and readily accessible to authorised users. Content must be *available* in terms of both uptime and access time. It should be possible for future OSNs to achieve the elusive "five nines" of uptime (99.999% uptime, or about five minutes of downtime per year). It should be possible for users to store content indefinitely, though perhaps at a cost (see problem "Cost" on page 20).

Users must also be able to access content quickly: storage should be accessible via caches or high-throughput Content Delivery Networks (CDNs), not dependent on the upload speeds of residential DSL lines. The time required to access content should scale linearly with the size of the content, not with the total size of all content in the system.

If a privacy-enabling OSN does not provide these availability properties, the majority of users who require and expect their OSN to "Just Work" will stay away. Users like Sophie who are willing to use the new network would therefore lose any benefits of cover traffic that might come from a large user base, discouraging even privacy-motivated individuals from adopting the service.

2.2.4 Untrusted infrastructure

In a privacy-enabling OSN, the providers of software and infrastructure should not be trusted. According to the definition of "Trust" on page 15, providers may be *relied on* to perform a service and provide availability, but they should not be be *trusted* to implement users' security policies. Integrity properties, including those provided by software, should be verifiable. Confidentiality claims should be grounded in something other than trust in a provider.

Solving this problem would allow a privacy-enabling OSN to change the outcome for the scenario subjects above who were upset by Facebook's decision to declare some private information as public (scenario "Facebook" on page 4). In a privacy-enabling OSN, it would not be within the power of the provider to make this policy change, since the provider is not trusted to enforce the user's confidentiality policy.

2.2.5 SOCIAL APPLICATIONS

Today's online social networks are not merely repositories of data: they are platforms that allow developers to create social applications. Future privacy-preserving OSNs should be designed with extension and adaptation in mind. As researchers such as boyd have shown, it will occur even if not planned for [58], so designers should provide application programming interfaces (APIs) that allow the OSN to be extended without compromising the privacy properties of the system.

Changing the outcome in scenario "Sophie" on page 5 requires that social applications be available on a privacy-enabling OSN, not just today's OSNs: otherwise, Sophie will be unwilling to use them. Developers should be able to write applications in well-known, widely-accepted languages. The platform that runs these social applications must provide applications with typical Operating System services such as reliable storage and inter-application communication. Applications must be able to manage their own namespaces of stored user data, but the security of user information should not depend on the quality of application code.

The platform's API should encourage developers to refer to personal information *à la* Felt and Evans' "privacy by proxy" [103]. For instance, Figure 2.1 on the next page shows two approaches to inserting the user's name in a UI element. It is preferable for an app to specify that "the opponent's name goes here" as in Listing 2.1 on the following page rather than require direct access to user data as in Listing 2.2 on the next page. Indirect use of user data should require no special permissions.

It may be possible for applications to request direct access to user data through a security API, especially if they compartment themselves into least-privileged components. Felt and Evans' survey of 150 Facebook applications found that only seven actually processed users' private information, performing functions such as choosing from a set of horoscopes based on the user's birthday [103]. On a privacy-enabling social application platform, a horoscope

Listing 2.1: High-level social abstractions.

```
ui.opponent.appendPlaceholder('opponent', 'name')
```

Listing 2.2: Low-level implementation.

```
for (var user : game.players()) {
  var name = user.name();
  if (name != my_name) {
    ui.opponent.appendChild(document.createTextNode(name));
    break;
  }
}
```

Figure 2.1: Direct and indirect access to user data.

application might still request direct access to the user's birthday for this purpose; such a request could certainly be honoured if the app actually ran as two applications, one with access to the birth date and one with access to the network, with a one-way information channel between them. Otherwise, user permission would be required. The path of least resistance, however, should be to refer to private information in abstract, indirect terms.

With user permission (see problem "Explicit expressions of user intent" on page 16), applications should be able to communicate with other applications, websites and users. Wherever possible, these privileges should derive from user actions rather than lists of permissions. For instance, rather than granting a "may open local files" permission when installing an application, applications should request access to local files via an Open File dialog that is rendered by the platform and only grants access to files that the user chooses.

Finally, the behaviour of social applications should be visible to users so that experienced users can examine application behaviour and critique it on behalf of the wider user community.

2.2.6 COST

Today's OSNs are nominally *gratis* — no direct financial obligation is placed on users — but they do have a cost. In order to pay for OSNs, user data is brokered, often indirectly but occasionally directly (see Chapter 3). Hundreds of millions seem willing to make this exchange of personal information for OSN services, but no alternative contract is being offered: no currently-available alternatives can offer the services of today's OSNs at a competitive scale.

Unlike in today's OSNs, the cost of providing a privacy-enabling OSN should be exposed to users. That cost might be paid in any number of ways — advertising, premium text messages, direct payment — but it should be low and it must not be hidden behind the exploitation of personal information. The platform should provide a payment interface that allows the use of arbitrary settlement mechanisms, creating a secondary market for privacy-preserving advertising — proposed independently by Bilenko et al. [48], Guha et al. [126], Pearce et al. [196] and Toubiana et al. [224] — or Chaum's anonymous e-cash [71].

The direct payment option might be particularly attractive to corporations, non-governmental organisations, etc. that wish to accrue the benefits of a shared data and application platform without the confidentiality costs of today's OSNs.

As in problem "High availability" on page 18, a solution to this problem would allow a privacy-enabling OSN to be practical for the current ranks of OSN users who, unlike "Sophie", have chosen practical systems over privacy-enabling ones.

2.2.7 LINKABILITY AND ANONYMITY

A practical, performant and scalable application platform may not be immune to traffic analysis: a malicious storage provider might link some possibly-encrypted data with client IP addresses. However, the system should be *compatible* with anonymity for users who wish to pay a cost: those who are willing to suffer higher latencies and lower bitrates should be able to communicate in ways that defy traffic analysis.

If a privacy-enabling OSN solved this problem, it would allow even "Sophie" to use the OSN for sensitive communications.

I have claimed in Section 1.2 that Footlights addresses all of these problems. Before I describe that system, however, I will first show that the existing research literature does not fully address these problems.

2.3 RELATED WORK

This section is not the only discussion of related work in this dissertation: each technical chapter contains a more detailed survey of work related to its specific remit. This section provides a high-level overview of systems that have attempted to solve privacy problems in online social networks.

The research literature contains many proposals to address the privacy problem in online social networks. There are new architectures: some encrypt user data within existing, centralised OSNs (§2.3.1) and some distribute user data over decentralised, peer-to-peer networks (§2.3.2). Despite some technical merit, these architectures fail to entirely solve the problems outlined in Section 2.2. Other proposals are apparently simply oblivious to the larger socio-technical context, providing technical solutions to entirely the wrong problems (§2.3.3). The literature also contains proposals to improve the user experience of privacy policy specification (§2.3.4). Further work in this area could allow user intent to be better captured as a formal policy (problem "Explicit expressions of user intent" on page 16), but policy alone does not provide privacy: mechanism is also required.

2.3.1 ENCRYPTION WITHIN OSNS

Over the past three years, there have been several proposals to add information hiding to existing OSNs, maintaining ties to large extant user bases while keeping user data hidden from OSN operators, advertisers and application developers. The most prominent feature of these proposals, the ability to interoperate with and be embedded within an existing OSN, is also their tragic flaw. A service so tightly coupled to extant OSNs cannot protect the social graph: the graph of users and the relationships among them.

The first proposal of this kind is Lucas and Borisov's flyByNight [165], which performs public-key cryptography in JavaScript, translating messages and other user data into opaque ciphertext that can be embedded in the usual social channels. In flyByNight, private keys are stored within the OSN itself for usability reasons, so a curious OSN operator could decrypt the traffic but other users, search engines and advertisers could not. A similar approach, which leverages the key management infrastructure of PGP, is Beato, Kohlweiss and Wouters' Scramble! [37]. Since Scramble! uses the PGP infrastructure, private keys are never visible to the OSN operator, but this system carries all of the usability problems inherent in PGP [231]. Other variations on this theme have been proposed based on Shamir secret sharing (Atrey et al. [33]), attribute-based encryption (Braghin et al. [61]) and group cryptosystems (Zhu et al. [247]).

Encryption-based approaches to information hiding carry the risk that the OSN operator might enact a "no blobs of ciphertext" policy. Luo, Xie and Hengartner's FaceCloak [167] avoids this particular risk by populating the user's Facebook profile with plausible-looking data, while keeping real information on another server that only FaceCloak users are aware of. Lockr, by Tootoonchian et al. [223] uses social relationships defined by OSNs to control access to information stored outside the network: users share "social attestations", tokens that represent a relationship, via a social network, and produce attestations to content servers which check them against "Social ACLs". Integration with services such as Flickr involve placeholders and external content storage services like FaceCloak. Guha, Tang and Francis' NOYB [127] uses the clever approach of storing real data, in plaintext, on the profiles of other NOYB users; the location of a user's actual data is given by a keyed permutation. NOYB does not address the key management problem.

The weakness common to all of these approaches is that they protect user data without addressing the most important problem: protecting the social graph. As I will show in Section 3.2, it does not matter if one hides their love of classical music from an untrustworthy OSN: the musical preferences of others in the social graph give them away [157, 181, 237, 246]. Similarly, encrypting a profile photo provides no protection from Jagatic's *social phishing* attack, in which information about the social graph is used to increase the effectiveness of phishing by making it appear to come from friends [136]. In such an attack, what matters is that some of the target's friends have made their names and photos available to the phisher. In contrast to schemes that achieve differential privacy [96], encrypting one's own profile data in an OSN does almost nothing to shield them from this attack; risks are produced by the actions of others.

2.3.2 Distributed social networks

There have been several proposals in the research literature for distributed online social networks that do not trust or rely on a centralised service provider, but none fully address all of the technical problems in Section 2.2.

HelloWorld was a 2009 proposal by Ackerman et al. to build a distributed social network in which users sent encrypted messages to each other over e-mail or Freenet [76]. User identities in HelloWorld consisted of mappings from a global namespace on `helloworld-network.org` to user-uploaded public keys. This architecture both *relied on* public infrastructure such as e-mail servers to perform their functions correctly and also *trusted* them not to observe the social graph as encrypted messages traversed them.

A more mature proposal for e-mail–based social networking is Fescher et al.'s "Mr Privacy" [105]. This system uses e-mail as a transport mechanism to support collaborative social applications. Like HelloWorld, protection of the social graph is based on trust in the underlying e-mail provider.

A related proposal is Yong et al.'s Mailbook [242]. Mailbook uses e-mail servers to store content that is shared with other users via a peer-to-peer meta-data layer. This combines the observability of centralised systems with the unreliability of P2P networks: the mail server can observe all accesses to content and the P2P network in question has been found to be unreliable by Buchegger et al. [65].

Mannan and van Oorschot have proposed a Kerberos-like system that uses instant messenger services to distribute capabilities for accessing private Web pages [168]. This allows user content to be distributed across many servers, but as in the above mail-based systems, the central IM server is not just relied on but trusted with the complete social graph. This approach seems to place no more trust in the IM server than in general IM usage, but this is not actually the case. The question to ask is not, "do you trust your IM server with your contacts?" but "would you continue to trust your IM server with your personal details if it became the gatekeeper for all of your online social interaction?"

Persona [36] uses attribute-based encryption to protect user data. This allows users to set up friend-of-friend groups, i.e. Alice can encrypt a message to "anyone that Bob has labelled with the attributes `friend` and `Cambridge`", but it is orders of magnitude slower than existing public-key techniques. For this multiplicative increase in computational cost, it provides no discernible security improvements. For instance, an example given by the Persona authors is that Alice can encrypt content for Bob's friends without Bob needing to entrust Alice with his list of friends. To take advantage of this affordance, however, Alice must send confidential information to a set of people whose membership she neither controls nor understands. Alice must trust Bob absolutely, since Bob can provide absolutely anyone with the `friend` attribute, or else Alice's message must not be very confidential. If the message is not very confidential, it is unclear why Alice would use a computationally-expensive encryption scheme to protect it rather than existing, efficient schemes.

Buchegger et al.'s PeerSoN [65] is a distributed social network based on a Distributed Hash Table (DHT) [201]. The authors rightly recognise that "the use of a DHT in itself [...] does not provide any security or privacy to the users"; the work initially concentrated on functionality rather than security, although broadcast encryption has subsequently been proposed as a mechanism to provide confidentiality properties [52]. At the time of writing, PeerSoN has no security implementation, so it remains to be seen whether a simple layering of encryption on top of PeerSoN's P2P network will satisfy the privacy problems in Section 2.2. The authors claim that PeerSoN will "utilize social trust — which is not present in most traditional DHTs — to make the DHT reliable" [65], but have not described how this will be done without revealing social graph information.

Safebook [79, 80, 81] provides access control via a set of nested rings called a *matryoshka* within an overlay network above a peer-to-peer overlay network. These rings consist of concentric circles of friendship relations: close friends, friends of close friends, etc. In order to access a user's content, it is necessary to find a path through this matryoshka to a node in the network that holds the data in question. This approach suffers from two problems: a lack of availability and a lack of a clear security model. On availability, churn in the underlying P2P network increases the difficulty of finding a suitable path through the matryoska. Cutillo et al. define "available" user data as that which can be accessed via the system with 90% probability; this falls well short of the commercial standards of centralised OSNs. On security, the matryoska system embeds unwarranted assumptions about trust. For instance, it assumes that each user will have a set of friends who are trusted with all information — this conflicts with problem "Multi-faceted identity" on page 17. Safebook also assumes that friends-of-friends are more trustworthy than friends-of-friends-of-friends, etc. Also, since Safebook stores protected data on friends' computers, the trust relationship is not simply that "I believe this person to be trustworthy" but also "I believe this person to be a competent system administrator who will keep their trusted node free of malware". The correct operation of the P2P overlay requires a "Trusted Identification Service" to guard against Sybil attacks [93] and impersonation. The authors claim that "this does not contrast [sic] our goal of privacy preservation through decentralization" [81] because "this service's jusrisdiction [sic] is limited to the purpose of authentication" [80], but the TIS is capable of violating the policy that "only trusted contacts of a node are able to link" that node's user ID and P2P node ID. Either this policy is important or it is not; the authors of Safebook want to have it both ways. This hazy trust model provides no clear basis for reasoning about security properties. Finally, Safebook provides no application model to allow third-party extension as required by problem "Social applications" on page 19: the authors refer to the system itself as a "social networking application" [81].

Aiello and Ruffo's LotusNet [28, 27] is based on the Likir variant of the Kademlia DHT. In this variant, security properties are claimed based on a binding between DHT nodes and persistent identities, verified by a trusted Certification Service. Although LotusNet is called a distributed OSN, it is dependent on global identities and the trusted global name authority for correct operation at its lowest levels.

The open-source Diaspora* project [276] is a federated social network. Users can create "pods", each of which hosts some number of users like a conventional, centralised OSN would. These pods can communicate to provide functionality at a larger scale. This architecture changes the parameters of trust somewhat, since users can choose which pod to trust with their data, but a pod must still be trusted just as a centralised OSN is trusted today. Diaspora* does not currently provide an application API: all code is curated by a centralised team of developers.

Narayanan et al. criticise distributed approaches to “personal data architectures” based on technical, economic and cognitive grounds [187]. The technical arguments are that distributed systems are more challenging to build than centralised ones and that decentralisation requires standardisation, which is a challenging process. These arguments are based on true premises, but neither need preclude the design of distributed social networks — they only preclude naïve designs. The authors also point out that economies of scale are important and switching costs are real, leading to the valid point of guidance that new OSN architectures should take economic considerations into account. Finally, the authors conflate several orthogonal cognitive considerations in social networks generally, describing them as problems with “decentralised systems”. For instance, the authors state that “more control over personal data almost inevitably translates to more decisions, which leads to cognitive overload”, but this is not a reason to avoid decentralised architectures. Centralised services such as Facebook provide very detailed and fine-grained control over some personal data, capable of causing just as much cognitive overload as an equivalent decision matrix in a decentralised system. What Facebook does not have is an incentive to explore alternative schemes for privacy policy configuration [13, 14] if those schemes hinder a “free flow of information” within the network [310]. In contrast, a distributed system that does not extract monetary value from user data has every incentive to combat cognitive overload by accurately capturing user intent, reducing decision fatigue and improving the user experience of privacy management. While the authors of this critique raise some important issues, their case against decentralised data architectures is greatly overstated.

Distributed architectures for online social networks could overcome the challenges described by Narayanan et al., but none of the current approaches solve all of the technical problems described in Section 2.2.

2.3.3 Solutions out of context

The encryption- and P2P-based approaches described above do not satisfy all of the requirements of Section 2.2, but many of these approaches do have a coherent model of the OSNs that they are meant to supplement or the larger socio-economic context that they are meant to operate in. Not all proposals in the research literature show evidence of this contextual understanding.

An example of a technical solution that fails to consider its social context is Maximilien et al.'s "Privacy as a Service" architecture [169]. This system is a centralised reference monitor [160] that users control by declaring a privacy policy with manually-assigned "privacy levels" for individual user data. Conceptually, this is exactly what Facebook *already does*: the difference is that Facebook does not make a "privacy risk index" salient to users via a "Privacy-aware Marketplace". The authors appear to consider the problem of privacy in OSNs to be solved, discussing in their future work section how their algorithmic complexity might be improved, not how users might overcome Whitten's *secondary goal property* — privacy and security are typically not a user's primary reason for using a system [230]. This is, how can users be convinced to spend their time manually assigning "privacy levels" to content? The authors do not seem to consider that the lack of privacy salience in today's OSNs may not be due to a lack of inventiveness or technical ability on the part of OSN providers but because of a desire for a "free flow of information" [310].

Similarly, Li, Alessio and Zhou's 2010 OST (Online Social Trust) model attempts to fit numerical trust models from OSNs onto CDNs in preparation for a coming convergence of the two [156], apparently oblivious to the fact that Facebook made a business decision several years ago to use CDNs with absolutely no access control, as illustrated by the 2009 study that Joseph Bonneau, Andrew Lewis, Frank Stajano and I performed on photo remanence in social network CDNs [20].

2.3.4 PRIVACY POLICY SPECIFICATION

Several works in the research literature attempt to improve user control of information sharing through improvements to the user experience of privacy policy specification. These proposals are independent of the underlying policy enforcement mechanism: they could apply equally well to any reference monitor [160] driven by a user-specified security policy.

Besmer et al. performed a user study in which Facebook installation dialogues were supplemented with additional information about the application's behaviour and the installation decisions taken by their Facebook friends [46]. Instead of the current "take-it-or-leave-it" model, users were permitted to specify what information would be shared with an application. The authors found that "motivated" users did modify their behaviour based on the extra information, but others did not.

In 2009, I co-authored a poster with Luke Church and Joseph Bonneau that proposed end-user programming techniques for the privacy policy specification problem [14]. In particular, we allowed users to create their own abstractions around subjects and objects, which could be expressed in natural language called "Policy Stories". In a preliminary study, our two users expressed confidence in their understanding of what they had specified using the system.

Other approaches to improving the user experience of policy specification include selectable, expert-supplied defaults [13] and machine learning of preferences to inform new policies generated on demand [101]. These approaches have the potential to aid users in clearly expressing their privacy preferences, informing the OSN "reference monitor" what should be shared. This approach is independent of the sharing mechanism.

These techniques could contribute to a larger privacy solution for online social networks, but policy alone does not solve problem "Explicit expressions of user intent" on page 16: a mechanism to enforce the policy is also required.

3

ANTISOCIAL NETWORKS

People have really gotten comfortable not only sharing more information and different kinds, but more openly and with more people. That social norm is just something that has evolved over time.

Mark Zuckerberg, January 2010 [350]

Today's centralised online social networks (OSNs) are trusted to make access control decisions on behalf of users, but those decisions do not always align with users' intentions. In some cases, technical flaws lead to data leakage contrary to the wishes of both users and operators. In other cases, there is a mismatch between user understanding of privacy settings and the actual protections afforded by the system. In many cases, however, the interests of the user and the network are simply not aligned. In these cases, when users' data must be safeguarded by operators without the incentive to do so, private information has a history of leaking out.

In this chapter, I consider how extant OSNs have revealed a great deal of their users' private information to third parties, often without permission (Section 3.1), and how, in some cases, sites may be revealing more than they themselves intend (Section 3.2).

3.1 USER DATA PRIVACY

Online social networks can be used to share information with friends, but the OSNs themselves can also share information beyond users' intentions. This disclosure has occurred through regular churn in OSNs' privacy settings

(§3.1.1), explicit leakage of personal information due to less-than-vigorous protection around advertising (§3.1.2) and by providing third-party applications with access to information that users within the network cannot see (§3.1.3).

3.1.1 Forced open: evolving defaults in privacy settings

Much unintended disclosure in OSNs stems from permissive, ever-changing default privacy settings. Default settings are important for two reasons. First, as shown in §3.1.1.1, many users never customise their privacy settings, so their security policy is exactly what the default settings dictate. Second changes in default settings over time can affect even those users who have specified a non-default sharing policy: users' expressed intentions may be disregarded as new defaults are created. §3.1.1.2 will show that when some new features were introduced to world's most popular OSN, all users were given the same default settings, whether those users were previously very private or very public. Even more strikingly, as the language used to express privacy policies changed over time, user-customised settings were lost and replaced with new, default settings. After these substitutions, a user who once expressed a very private sharing policy could end up with a quite permissive policy.

3.1.1.1 Use of default settings

OSN users have, over time, become increasingly willing to spend time adjusting privacy settings to match their sharing intentions. Nonetheless, default settings are still used by many users today.

Gross, Acquisti and Heinz performed a study in 2005, the early days of Facebook, and found that within a university "network" — a Facebook mechanism for separating users from different institutions — users engaged in very high rates of personal information disclosure [125]. 90.8% of user profiles displayed a photo to all Facebook users in the university, 87.8% displayed a birth date, 50.8% a current residence and 39.9% a phone number. Facebook profiles contained, by a small margin, more personally identifiable information than the competing, less exclusive OSN Friendster. In a later study, Acquisti and Gross found that 77% of users had no desire to change the default searchability settings [26]. Gross, Acquisti and Heinz speculated that exclusivity in an OSN is inversely proportional to voluntary disclosure.

This speculation was borne out in a 2008 study by Krishnamurthy and Wills [147], which found that profile visibility in Facebook "networks" varied from 51% to 93%, depending on network size. In large networks, users were more likely to hide their profile information, whereas in smaller networks, users were most often content to leave their profile at the default visibility. These numbers contrast with the statistics that Krishnamurthy and Wills found for Twitter, a personal broadcast medium, in which over 99% of users employed default (public) privacy settings.

A 2010 survey study by boyd and Hargittai [59] found that user interaction with Facebook's privacy settings increased dramatically between 2009 and 2010, but the majority of "occasional" Facebook users have interacted with their privacy settings three or fewer times and only 53% of "frequent" users had interacted with their settings four or more times.

Based on these results, it is reasonable to assume that more and more OSN users will modify their privacy settings over time, customising them from the OSN-provided defaults to settings that better meet their privacy and sharing goals. Nonetheless, it also seems prudent to assume that there will always be a population of users who keep the default settings.

Still, the fact that many users express their privacy and sharing intent within an OSN does not completely insulate them from future disclosure: if the OSN changes its default settings or the language that privacy settings are expressed in, actual patterns of disclosure may not match users' intent.

3.1.1.2 A HISTORY OF CHANGING DEFAULTS

In today's OSNs, when a user modifies her privacy settings, expressing her intent about who she wishes to share with, she can only express policies in a language made available by the OSN. This language is typically a list of rules that state "items of class X may be shared with users of class Y", where X might be "all photos" or "this photo album" and Y might be a friend list or an individual. A user may be able to express the policy "only share photos from the party last weekend with people I have explicitly designated", but she cannot control how her name will be used by a not-yet-developed OSN feature: the language affords no means of expressing it. An OSN *could* provide the vocabulary to express the concept “and any other use which I have not explicitly described”, but that has not been the practice of today’s OSNs. On the contrary, as I will demonstrate in this section, the world’s most popular OSN tends to roll out new features with mandatory information disclosure, then provide an opt-out facility after user outcry.

Even if users rigorously restrict their current privacy settings, a steady stream of new features provides a corresponding stream of new switches to be turned off, and most of these switches default to “on”. An OSN account at rest, with no forces acting on it other than the OSN itself, tends to become more publicly-accessible over time.

I will illustrate this tendency by considering the case of Facebook, summarized in Figure 3.1 on the facing page. This figure shows a timeline of Facebook’s development, including new features, privacy-related developments, newsworthy events and public statements.

Facebook, which is currently the world’s most popular OSN, opened to the general public on 26 Sep 2006 [284]. Prior to that, users were required to have a verifiable e-mail address from a “network” such as a university. Suppose a University of Cambridge student named Alice created a Facebook account in August 2006 and immediately limited the sharing of all personal information to direct “friends” only, but left her “searchability” setting at “Everyone” so that friends could search for her and send friend requests. Without this setting, it was very difficult for real-life friends to find each other on Facebook: it was impossible for two users with the most restrictive searchability settings to “friend” each other, which discouraged users from choosing such settings.

Here is a history of how Facebook’s changing default settings would have affected Alice’s sharing policy from then up to the time of writing, assuming that she made no further changes to her own privacy settings:

15 Aug 2006 The Facebook development platform is introduced [286], allowing users to install applications which access their private data and provide services such as social games. Alice’s friend Bob installs an application, granting it access to his profile and any information shared with Bob by

Privacy			Features	Other
Positive	Negative			
	News Feed, Mini-Feed (insufficient control)	2006	FB mobile My Shares External Shares First Friend Lists Platform Beta	Open to General Public
My Privacy page Beacon becomes opt-in	Public Search Listings (opt-out) Social Ads, Beacon	2007	FQL Subscriptions Platform Beta FB Message to e-mail Friend Lists	
Standard in-line privacy controls, restricted sharing	Facebook Connect	2008	Facebook Chat External event import Friend suggestions	
Regional network phase-out Per-post privacy controls Beacon shuts down	"Like" button (always public) PII leakage paper Apps can request e-mail Public photo, friends, etc.	2009	Facebook usernames	Governance vote Canadian Privacy Commissioner investigations
Regional network phase-out Privacy controls on app content Non-public connections, opt-out from Platform and IP App permissions request Mobile privacy controls Closed-by-default groups	Automatic Pages connections, "Like" everywhere, Instant Personalization WSJ PII leakage report Places (friends check *you* in) Rapleaf scandal	2010	Facebook Chat everywhere Check-in Deals Android SSO	
HTTPS, Social Authentication Profile review	Sponsored Stories Timeline	2011	New Deals Automatic face tagging Smart Lists Subscribe button Friend Activity	Google smear campaign

Figure 3.1: Facebook privacy timeline

his friends, including Alice. Whatever personal details Alice has shared with Bob are sent to an application server run by the application's maintainer, rather than Facebook. This data is public according to the definition of "Public" on page 15: no technical restrictions are imposed by Facebook on what the application maintainer can do with Alice's data.

5 Sep 2006 News Feed and Mini-Feed are introduced [301]: information that was previously available to users who looked for it (e.g. "Alice is not in a relationship") is now broadcast to them ("Alice is no longer in a relationship with Charlie"). Users protest, leading founder Mark Zuckerberg to declare that the incident is just a misunderstanding: no privacy settings have changed [311]. Three days later, he admits that the switch from accessible information to broadcasted information is, in fact, a significant change; new privacy controls are both required and available, but on an opt-in basis [310].

5 Sep 2007 Public Search Listings are introduced [287]. People who are not signed in to Facebook — as well as search engine robots — can now view a page containing selected information about Alice such as her name, photo and a subset of her friends. This information both disambiguates Alice from other Alices on Facebook and shows external viewers that this particular Alice and her friends are using Facebook.

6 Nov 2007 Facebook Pages and Social Ads are introduced [297]. Brands acquire the ability to create pages that look like user profiles, but rather than "friending" a brand, users can declare that they are a "fan" of one. This information is then used in Social Ads, where users' names and likenesses are used to advertise the brand in question to their friends, e.g. "Alice is a fan of Brand X" or "Bob wrote a review of Local Restaurant Y".

9 Nov 2007 Beacon is introduced [300]. The actions of logged-in Facebook users are tracked on external websites and announced within Facebook, e.g. "Alice just bought *Security for Ubiquitous Computing* by Frank Stajano from Amazon.com". User outcry leads Mark Zuckerberg to apologise, both for making Beacon opt-out rather than opt-in and for failing to respond to user complaints in a timely way [312]. Beacon becomes opt-in, then two years later, is shut down entirely as part of a legal settlement [281].

27 Apr 2009 The Open Stream API is released [292]. Alice's favourite newspaper, which requires readers to log in before they can read all stories or post comments, starts using Facebook Connect for authentication. Alice now logs into her news site by clicking the Facebook Connect button rather than entering a username and password: she appreciates the opportunity to remember one less password. With the Open Stream API, that news site can now publish items in her Facebook News Feed such as "Alice just commented on story *X*" without Alice's permission: to stop this behaviour, Alice must explicitly opt out.

2 Jun 2009 Facebook begins phasing out regional networks [295]. Events and groups that were once restricted to a regional network are now open to everyone, and Alice's profile acquires a "Current City" field, which is automatically populated as "Cambridge".

29 Oct 2009 Applications (including sites using Facebook Connect for authentication) are now able to require that Alice provide her e-mail address in order to use their services [289]. Since Alice does not want to lose all of the comments, history and other content which she has built up on her favourite newspaper's website, she clicks "OK".

9 Dec 2009 While announcing "new tools to give you even greater control over the information you share", Facebook announces that "a limited set of basic information that helps your friends find you will be made publicly available". Facebook publicly discloses Alice's name, photo, networks and her previously-private friend list [302]. Alice's list of Facebook friends is now visible to all Internet users and search engines.

10 Dec 2009 Facebook provides a mechanism for users to opt out of having their friend lists displayed publicly [296]. Without active intervention, Alice's friend list — which was once hidden even from her friends — remains public. Bob opts out of the publicly-viewable friend list, but the setting does not have the effect he expected: his friend list is no longer displayed on his Facebook profile page, but the list itself "is still publicly available, however, and can be accessed by applications" [296].

19 Apr 2010 Some textual profile fields become "connections" to new Facebook Pages [294]. The next time Alice logs in, she is prompted with a dialog asking if she wants to link to the University of Cambridge page, since her university affiliation was automatically inserted into her profile when network-based access control was deprecated. If she clicks the default button, thinking "that's accurate information" rather than "I want this information to be shared publicly", her university affiliation will become a "Connection", considered "public information" that cannot be hidden by any privacy setting.

21 Apr 2010 Application policy is changed to permit applications storing user data indefinitely (up from 24 h) and using it "in any way you believe provides a richer experience for your users" in order for application developers to "build a closer relationship" with users [285]. Websites that Alice has logged into with her Facebook credentials and required her e-mail address as a condition of use begin sending her daily e-mails that Facebook can neither monitor nor stop.

The same day, the "Like" button goes global: actions taken on external websites, such as clicking a "Like" button, appear in the internal Facebook News Feed [291]. Echoes of Beacon ring clearest when Alice makes a comment on a news website, in the context of a discussion around a particular article, and it appears in the Facebook News Feeds of people Alice does not know. This is later restricted to the feeds of Alice's friends [290], most of whom were not part of the original discussion's context.

Instant Personalisation is also announced, though it is not yet known by this name [313]. When Alice visits the review site Yelp, it is informed by Facebook that Alice is visiting and shown her connection to the University of Cambridge, as well as connections to any local restaurants that Alice has already "liked". Later, Alice views a PDF file containing membership rules for a Cambridge society on the document sharing site Scribd. As a trusted Instant Personalisation partner which has been "carefully chosen, reviewed and ... contractually required to respect people's privacy preferences" [290], Scribd has immediate access to Alice's friend list, which it uses to acquire Bob's e-mail address and sign him up for an account with no clear opt-out mechanism, informing him that Alice has signed him up [347].

26 May 2010 Facebook no longer requires all connections to Pages to be public and provides opt-out facilities for both the Facebook Platform and Instant Personalisation [315]. As a casual user who does not follow Facebook's press releases Alice is unaware of the opt-out mechanism. Alice's connections continue to be publicly-visible, and Facebook's Instant Personalisation features continue to provide her profile details to partner websites.

30 Jun 2010 Facebook launches a new application authorisation process [305]. As Facebook had promised the Canadian Office of the Privacy Commissioner in August 2009 [270, 271], a dialogue box appears when applications are installed to inform the user what personal information the application requires and to present "Allow" and "Don't Allow" options. Controls are also provided in Alice's privacy settings page to restrict what information Bob and other friends can disclose to applications on her behalf, should she be aware of and choose to actively manage them.

19 Aug 2010 The Places feature is introduced [303]. Users now have the ability to "check in" (register their position) at real-world locations. By default, check-ins are visible to all friends. Alice never checks in to Places herself, but when she and Bob visit a local coffee shop together, Bob checks in and declares that he is with Alice; "it is as if [she has] checked in at that place [herself]" [303]. The first time this happens, Alice is asked for authorisation. Because of implied social pressure and because it seems innocuous, Alice chooses the default option ("Allow"). From then on, no authorisation is required for such check-ins unless Alice opts out of the service.

6 Oct 2010 Facebook provides closed-by-default groups and an application dashboard to illuminate what applications are authorised to access Facebook information [314]. By clicking an "Edit" link, users can also see what information applications require and what is optional.

13 Oct 2010 Facebook users' names and likenesses start appearing in Microsoft's Bing search results [306]. Now, when Bob searches on Bing, he will see Alice's name and photo endorsing products or websites that Alice has expressed a "liking" for.

25 Jan 2011 Facebook introduces Sponsored Stories [274, 326], in which advertisers can pay for Alice's activities, already published in her News Feed, to be promoted as advertisements. When Bob visits a local coffee shop with Alice and other friends, his "check-in" declaring their real-world-position may be used as an advertisement for that coffee shop and shown to Alice's friends. This is a paid-for advertising feature; no opt-out is provided.

26 Jan 2011 Facebook introduces opt-in HTTPS for all Facebook content as well as "Social Authentication", in which users coming from unusual IP addresses are required to identify faces of their friends while logging in [299]. Users may now opt in to Transport Layer Security (TLS) protection of their Facebook sessions, but Alice, using the default settings, is still using Facebook unencrypted in her local Cambridge coffee shop.

In 2006, Alice configured her Facebook privacy settings to be restricted but still practical. If she then left them alone and accepted whatever default "Allow" buttons she was presented with, she would hardly recognise her privacy settings five years later. Information that she once declared private, such as the list of her friends, would now be visible to the world and indexed by search engines. Her name and likeness would appear in advertising, not just on Facebook itself, but next to products and websites in Bing search results. Her activities, such as commenting on a story on a newspaper's website, would appear as advertisements within Facebook, as would the fact that she visited a popular local coffee shop yesterday, even though she herself did not tell Facebook that she was there. Her friends' details have been relayed to "partner" websites, which have signed them up to services and mailing lists that they have no interest in and that are difficult to opt out of. Facebook provides more privacy controls today than it did in 2006. By default, however, Alice does not benefit from them.

In summary, changing defaults in OSNs are not a matter of purely academic interest. When a user signs up for an OSN, they enter into an agreement in which the user provides sensitive personal information and the OSN protects that information according to a sharing policy specified by the user. When the OSN changes the policy's vocabulary, without revisiting the policy or even providing notice that the vocabulary has changed, the user's original intent can be lost. Alice set a restrictive policy in the 2006 vocabulary. In the 2012 vocabulary, however, she is as open as Facebook wants her to be.

3.1.2 FAILED OPEN: ADVERTISERS AND PRIVATE INFORMATION

Personal information can leak from social networks when users fail to explicitly forbid it via opt-out mechanisms, but this is not the only leakage channel. The private information of users — OSNs' *product* — has been explicitly leaked to advertisers — OSNs' paying *customers* [342].

In 2009, Krishnamurthy et al. [148] identified the leakage of Personally Identifiable Information (PII) from several online social networks via request URIs, referrers, etc.. Some of this leakage may have resulted from simple carelessness. For instance, at the time of Krishnamurthy et al.'s study, Facebook used user IDs embedded in the URI query string, so if users clicked a link to an external site, their UID would be exposed in the HTTP Referrer header. This has since been rectified so that, as of September 2011, a user who clicks a Facebook advertisement will present an opaque binary Referrer header to the target site.

Other instances of PII leakage are more more overtly intentional: Krishnamurthy et al. discovered that some OSNs explicitly encoded user age, gender and even postal code in advertising request URIs. Even more egregious are the third-party applications described in June 2009 by Bonneau [343] that encoded user ID, user name, profile photo *and a list of friends* in advertising request URIs.

In July 2010, Korolova demonstrated [145, 146] that Facebook's advertising platform can be abused to learn private information about users with very restrictive privacy settings. Any information uploaded to Facebook may be used to select which advertisements are shown to users; users cannot use privacy settings to control Facebook advertising. Korolova found that advertisers could specify such precise targeting that the ad will only be shown to one Facebook user, often using only publicly-available information. Facebook then reported "aggregate" statistics, such as impression count, based on that one user. An attacker can target an ad to one user and then vary a parameter of interest (e.g. age, birthday, sexual orientation), watching to see which values generate impressions and which do not. Facebook released a fix within a week of notification, namely to block ads which do not reach at least 20 users, but such a simplistic defence can easily be defeated by the tracker attack, introduced in 1979 by Denning et al. [88] or — even more simply — by creating 20 accounts that match the targeted characteristics of the query subject [146].

No matter what is claimed by vocal elements of the press [277, 278], Facebook does not directly sell user data to advertisers. Others OSNs have, however, and a lack of technical protection around user data has allowed third-party application developers to sell user data to advertisers. Weak protections also allow advertisers to learn users' private details via microtargeted advertisements.

3.1.3 Left open: application access to private user data

In this section, I will show how private information has also been shared with third-party OSN applications, both by surveying the work of others and by describing research performed by myself and colleagues at the University of Cambridge. We found that some applications can access more private information when they run than the user who they are nominally running on behalf of.

Today's OSNs provide rich platforms for third-party social applications. Application functionality is supported by private user information, often served in greater quantities than actually required. Safeguards against misuse rely heavily on legal rather than technical protection. Felt and Evans, in their 2008 study of 150 Facebook applications [103], found that 90% of applications had no need of private user data, but were being given it anyway. Several applications used information in ways that contravened Facebook's Terms of Service, making information visible to users who would normally be unable to view it.

Listing 3.1: Example Facebook Markup Language (FBML) tags — now deprecated.

```
<fb:name id="[$id]">    <!-- Name of specified user. -->
<fb:friend-selector>    <!-- Drop-down friend selector. -->
<fb:visible-to-friends>
    <!-- Visible only to friends of the user. -->
</fb:visible-to-friends>
```

Of the 150 applications studied, 141 accessed user data for the sole purpose of displaying it to the user. Felt and Evans proposed that OSNs adopt a *privacy-by-proxy* design pattern, providing named placeholders that applications can use to manipulate user information indirectly. For instance, an application might instruct the Facebook UI to insert a user's name in a particular place without the application learning the user's name. In fact, as the paper points out, Facebook had such a mechanism, called the Facebook Markup Language (FBML), which allowed developers to insert indirect references to user data as shown in Listing 3.1. Its use was optional, however, used purely as a performance optimisation to reduce the number of round-trip messages between application servers and Facebook. FBML was subsequently deprecated in in 2010 in favour of newer APIs [288] and was deactivated entirely in 2012 [298].

In 2009, Facebook was investigated by the Office of the Privacy Commissioner of Canada in response to a complaint, an unsatisfactory response and a second complaint [270]. Following the second investigation, Facebook agreed to provide more user control over application behaviour [271]. User consent is now sought before applications can obtain personal information: users are prompted with a dialogue that describes the information sought and provides an opportunity to cancel application installation. Such cancel-or-allow approaches have been rightfully criticised by Yee as "security by admonition" for "forc[ing] security and usability into conflict" [241] rather than learning the user's intent. Furthermore, once divulged, no technical safeguards exist to prevent misuse of private user information (see the definition of "Public" on page 15).

Facebook provides applications with programmatic access to user data via the Graph API [253] and the Facebook Query Language (FQL) [252, 304], which

Listing 3.2: A Facebook Graph API response.

```
{
    "id": "516792779",
    "name": "Jonathan Anderson",
    "first_name": "Jonathan",
    "last_name": "Anderson",
    "username": "jonathan.robert.anderson",
    "locale": "en_GB"
}
```

Listing 3.3: Sample FQL query: what friendships exist among two sets of users?

```
SELECT uid1,uid2 FROM friend WHERE uid1 IN (u1,u2,...) AND uid2 IN
    (u3,u4...)
```

is a restricted variant of the standard Structured Query Language (SQL). The Graph API is a Web service: applications use HTTP to retrieve data formatted with the JavaScript Object Notation (JSON) [258]. For example, the data visible at the URL `https://graph.facebook.com/jonathan.robert.anderson` is the public profile of the author, shown in Listing 3.2.

While writing collaborative papers with Joseph Bonneau, Ross Anderson and Frank Stajano in 2009 [7, 8], before the new application restrictions were introduced, I used FQL queries of the form shown in Listing 3.3 to exhaustively query the “friend” relations among 18,000 users in the orignal Harvard University user ID space (users with IDs below 35,647) and among 15,000 users in the early Stanford University ID space (between 200,001 and 225,615). We found that Facebook would not reply to requests involving very large user lists, but by limiting each user list to 1,000 users, we were able to exhaustively query the relevant spaces’ friend relations.

Later, approximately three months after announcing that an agreement had been reached with the Office of the Privacy Commissioner of Canada, Facebook announced that some user data, including the list of one’s Facebook friends, would become public information — users would no longer be able to hide their Facebook social connections [279]. The next day, amidst user complaints,

Listing 3.4: A simple FQL query: are two users friends?

```
SELECT uid1,uid2 FROM friend WHERE uid1=X AND uid2=Y
```

Facebook provided an option for users to make their list of friends appear private again [280]; users can conceal their list of friends from all other users, but friendship information is *not* kept hidden from third-party applications.

In September 2011, even with Facebook's new application restrictions in place, I confirmed that a Facebook application can still issue an FQL query to discover whether or not there is a "friend" relation between two specific users or among two sets of users. Using an FQL query very much like the example in Listing 3.4, I found that an application running with the credentials of a throw-away user account could ascertain whether or not there was a "friend" relation between any two users, even if both users have configured their privacy settings to keep their friend lists secret.

The risk of application misbehaviour is not purely academic: in October 2010, the Wall Street Journal discovered [282] that popular applications such as Zynga's FarmVille, which at the time had 59 million users, shared information about users and their friends with data brokers [308] such as Rapleaf [275]. These brokers build user profiles to link user identities across platforms and sell the data to customers such as online retailers who use it to personalise mass e-mails and other "marketing experiences". Rapleaf can furnish firms with customer information such as their family status and home market value for $.01 USD per e-mail address [316]. Other fields, such as consumer loan-to-value ratio, are no longer publicised on Rapleaf's website but are still accessible through the Rapleaf API, as shown in Listing 3.5 on the next page. At the time of the Wall Street Journal story, Rapleaf advertised OSN identifiers for sale [275], though their spokesman declared that "We do not sell Facebook IDs to ad networks".

In the days after the Wall Street Journal story broke, Facebook issued a series of statements [283, 308, 309] describing three new features. The first was a new UID encryption feature: Facebook UIDs were encrypted with the symmetric key of the application receiving the UID in order to mitigate the risk of

Listing 3.5: Response to a Rapleaf API query (pre-purchase) [319].

```
{
  "household_income": "Data Available",
  "loan_to_value_ratio": "Data Available",
  "invested_assets": "Data Available",
  "gender": "Male",
  "length_of_residence": "Data Available",
  "children": "Data Available"
}
```

accidental disclosure. Second, a new *third-party UID* was created to be used by applications that share unique identifiers with third parties. Third, an explicit prohibition was enacted on applications passing user information to data brokers [307]. These measures did nothing to prevent intentional disclosure by applications, however. In this case, Facebook punished several application developers, which it described as "fewer than a dozen, mostly small developers, none of which are in the top 10 applications on Facebook Platform" — at odds with the Wall Street Journal's allegations about Zynga, one of Facebook's largest third-party application developers. Facebook banned these developers from the Platform for six months, and required them to "submit their data practices to an audit in the future" [308]. Facebook continues to assert the safety of user information given to third parties, not based on technical protections but on contractual compliance. That is, according to the definition of "Public" on page 15, information shared with third-party application developers is public.

3.2 SOCIAL GRAPH PRIVACY

Explicit disclosure, whether discretionary permissive defaults or mandatory public information and advertiser access, is not the only way in which users' private information reaches the outside world. Specific data items about OSN users, such as personal interests and favourite cultural artefacts, is valuable to plausible attackers such as scammers. However, the social graph is also valuable, and OSNs may leak more information about it than they intend.

In many attacks, the confidentiality of a user's profile data is irrelevant. Jagatic's social phisher [136] does not need to know my favourite book or West End show, he needs the names and profile photos of my friends. Even when the private details of individual profiles are relevant, many of them — from political views to sexual orientation — can be probabilistically inferred from those of friends [157, 181, 237]. Finally, specific details about a user can almost always be obtained through a targeted attack, e.g. "spear phishing"; the difficulty for the attacker is in determining, at a very low unit cost, which potential target is worth the cost of a targeted attack, as demonstrated by Herley [131].

Because of this, it is insufficient for a social network to protect user data without protecting the social graph. In this section, I consider the protection of the social graph in the context of public search listings.

3.2.1 PUBLIC SEARCH LISTINGS

As described in §3.1.1, Facebook introduced Public Listings in September 2007 to "help more people connect and find value from Facebook" [287]. By exposing some user information to the public Internet, Facebook could encourage non-users to join the service. For instance, a non-user Nancy might see that her friend Bob uses Facebook, as do Bob's friends — some of whom might be mutual friends. This information might encourage Nancy to sign up for her own Facebook account.

Furthermore, by exposing Public Search Listings to search engines, people searching for Bob would find Bob's Facebook page, raising the profile of Facebook as a communication medium for first contact. Other OSNs such as LinkedIn are even more aggressive about search listings: whereas Facebook allows users to opt out of Public Search Listings, LinkedIn does not.

Publicly exposing all data held by a service would remove the incentive for non-users to join the OSN (to say nothing of the reaction of current users); the premise of search listings must therefore be that some limited information from the social graph can be exposed without revealing all user information. In this section, however, I demonstrate that, while it is possible to shield user details from search engines, exposing almost any information about the social graph allows adversaries to approximate important graph characteristics closely.

3.2.2 PUBLIC LISTINGS MODEL

In early 2009, Joseph Bonneau, Ross Anderson, Frank Stajano and I studied the problem of social graph privacy in the presence of limited disclosure [7]. We were motivated by, and modelled the problem on, Facebook's usage of public listings. As of January 2009, this feature provided a set of eight friends for each public listing — since then, public listings have exposed more data. The set of friends that is displayed seemed to be a function of the viewer's location: repeated queries from a Tor exit node [91] yielded the same eight friends, and the set of revealed friends depended on the geographic location of the node — the authors' public search listings contained primarily North American friends when querying from a North American exit node but more European friends when querying from a European exit node. Those in Cambridge, UK viewing a public search listing might be more swayed by the presence of UK users than Canadian users, since they are more likely to be mutual friends.

Public listings are intended to be indexed by search engines. We encountered no technical measures to limit the rate of queries: we retrieved approximately 250,000 public search listings per day from a desktop computer connected to JANET, the UK's Education and Research Network. This figure comes from a rather simplistic approach to crawling: we dispatched work to a few tens of crawlers from a single, contended relational database where all query results were stored. A focussed engineering effort would certainly yield better results, so it is safe to assume that the complete set of public search listings is available to motivated parties — or contractual partners. This has been demonstrated by Microsoft, which includes publicly-visible Facebook information in its Bing search results [306].

This section uses a model of public search listings depicted in Figure 3.2 on the next page. A social graph G consists of vertices V, representing OSN users, and edges E, representing one-way "friend" or "follower" relations among them. This graph is sampled to produce a graph $G_k =< V, E_k >$ which shares vertices V with G, but in which $E_k \subseteq E$ is a set of edges produced by randomly selecting k outgoing edges from each vertex $v \in V$.

We computed several functions that might be of interest to attackers over the

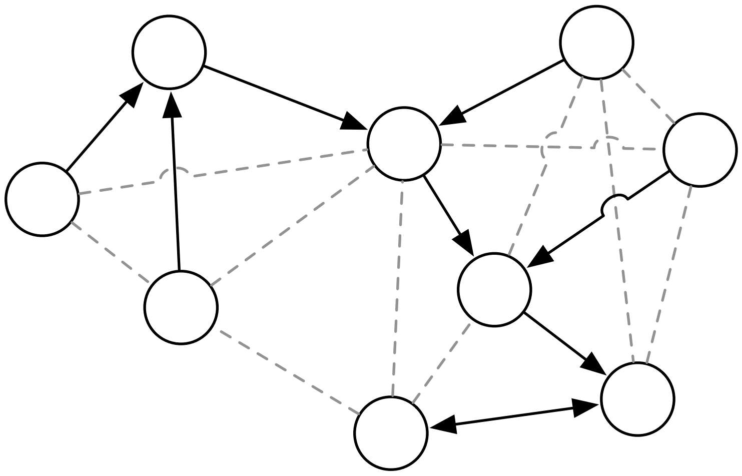


Figure 3.2: A sampled graph G_k with $k = 1$.

sampled graph, e.g. betweenness centrality, which identifies highly-connected users who are in a position to influence the spread of rumours and other messages (see §3.2.4.2). We compared the results to those that would be obtained from the complete graph in order to determine how close an approximation of these functions an attacker could develop from sampled public search listings.

Our model assumed that sampled edges are available for all vertices in the graph, that is, no users have opted out of public search listings. This assumption is based on an ad-hoc exploration at the time of the study, in which we found that fewer than 1% of users in the Cambridge network had opted out.

3.2.3 GRAPH DATA

In order to evaluate the effect of sampling on a graph's properties, we required a social graph that we could treat as ground truth and from which we could derive a sampled graph. Public search listings provide an already-sampled graph, and crawling within the Facebook network was ineffective — as shown in §3.1.1, not all users expose their profiles to non-friends. Instead, we built a Facebook application using the FQL feature described in §3.1.3 to identify users and friend relations. We derived our data from well-defined user ID spaces for universities with early access to Facebook. These Facebook networks, shown in Table 3.1, have contiguous, densely populated user ID spaces.

Table 3.1: Basic statistics from early Facebook networks.

Network	ID Space	Nodes	Edges	$\lvert E\rvert/\lvert V\rvert$
Harvard	0–35,647	18.2 k	1.06 M	58
Columbia	100,000–127,322	15.4 k	620 k	40
Stanford	200,000–225,615	15.0 k	945 k	63
Yale	300,000–317,759	10.5 k	639 k	61

Since our network-crawling application relied on FQL rather than public search listings, it was subject to rate-limiting. Friendship discovery among users required $O(n^2)$ queries, where n is the number of users in a network. Applications were not permitted to query complete friend lists for all users even in 2009, but rather had to employ the techniques shown in §3.1.3. Nonetheless, we were able to exhaustively search spaces of 25,000–35,000 user IDs for active Facebook accounts, and to query for friend relations among 10,000–18,000 active users in less than 12 h.

We performed our study before Facebook applications were subject to fine-grained privacy controls [293], so the only way to conceal particular data items from an application was to opt out of the Facebook Platform entirely. In 2009, this option was rarely taken. For instance, out of approximately 500 users known to exist in the Stanford user ID space, only three were not visible to our FQL-based application.

Our data sets only included relations between users within a UID range. That is, we captured relationships among users attending the same university, but not friendships with people outside the university. These social graphs are incomplete and therefore may not be representative of the complete network, but they do provide views into real social graphs rather than simulated data. We did not use them to characterise the social networks themselves, but to characterise an attacker's ability to approximate certain graph statistics when presented with a partial view of those graphs.

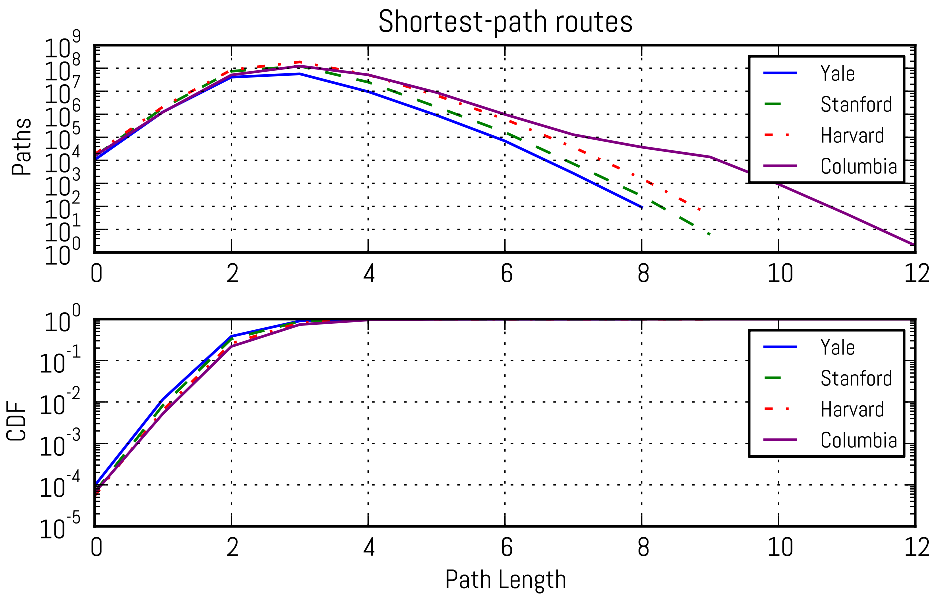


Figure 3.3: Route lengths and reachability in university networks.

3.2.4 Approximations of graph characteristics

In our study, we demonstrated how several important characteristics of a social graph can be approximated. In this dissertation, I have only included those characteristics for which I did the analysis work. This leads to a rather obvious omission — I do not discuss the approximation of users' degree. I do, however, discuss the reachability of and shortest paths between nodes (§3.2.4.1) as well as centrality and its potential use by attackers to influence messages in a social network (§3.2.4.2).

3.2.4.1 Reachability and shortest paths

A key characteristic of social networks is connectedness: can every person in the network reach every other person via short social paths? This connectedness can be measured via the lengths of the shortest paths through the network. If some nodes are disconnected from the rest of the network entirely, then some shortest-path routes will not exist at all: they have infinite length. A *connected* graph will have finite lengths for all routes.

The lengths of shortest-path routes through the four target networks are

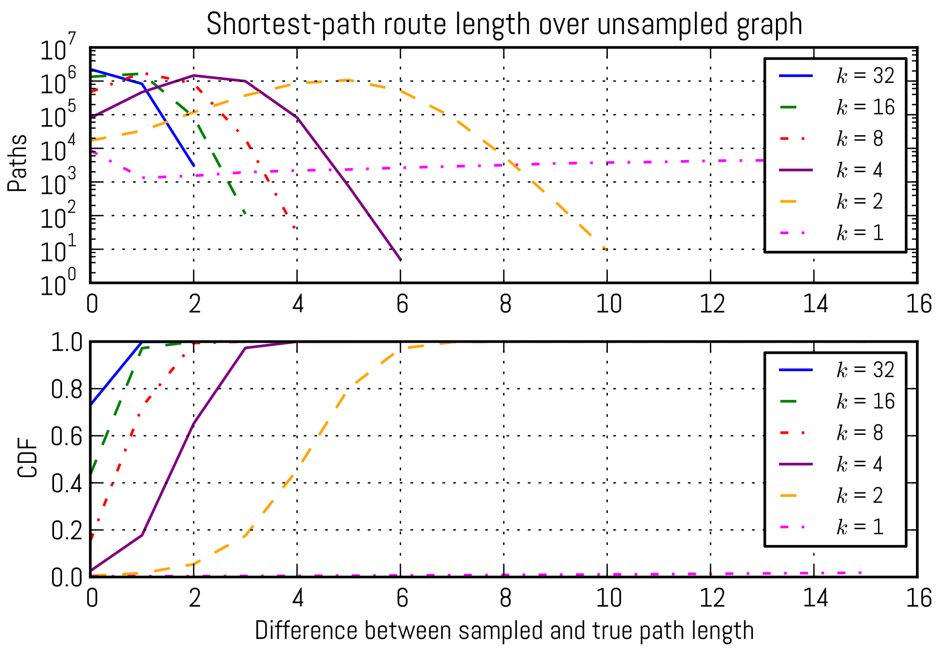


Figure 3.4: Route length approximation under k-sampling (Columbia).

shown in Figure 3.3 on the preceding page. This figure shows, for $l \in [0, 12]$, how many shortest-path routes of length l exist from any node i to any other node j. The top figure shows the total number of such (directed) paths, whereas the bottom figure shows the total number of routes of length l or less as a fraction of the total number of shortest routes $|V|^2$.

Figure 3.4 shows the effect of k-sampling, as described in §3.2.2, on the Columbia University network. This figure shows a frequency count of the possible differences in path length between true and sampled values. For all illustrated values of k, some paths have a non-zero difference in length between the true and sampled graphs. This effect is least pronounced in the $k = 32$ case, in which over 75% of paths have a difference of zero, but even then, some paths are longer than in the complete graph. A tabular summary of this information is shown in Table 3.2.

For $k > 1$, almost all shortest-path routes in the sampled graph are longer than the unsampled graph by a small integer number of hops. In the $k = 8$ case

Table 3.2: Increase in shortest-path route lengths caused by k-sampling.

k	Increase in route length						
	0	1	2	3	4	5	6
32	73.1%	99.9%	1				
16	43.5%	97.2%	99.996%	1			
8	**15.3%**	**72.3%**	**99.4%**	**99.99999%**	**1**		
4	2.59%	17.7%	65.2%	97.3%	99.98%	99.9998%	1
2	0.55%	1.63%	5.45%	17.5%	45.5%	80.2%	97.0%
1	0.28%	0.32%	0.37%	0.44%	0.51%	0.58%	0.67%

— the original motivation for this work — 99.4% of all shortest-path routes are lengthened by two hops or fewer and an attacker can approximate true route lengths by simply subtracting 1.

3.2.4.2 CENTRALITY AND INFLUENCE

Another important metric in social network analysis is *centrality*, which is a measure of the importance of members of the network based on their position in the graph. Social graph members with high centrality are in a position to wield influence disproportionate to their degree, because they are not merely connected to other nodes, they are *efficiently* connected. Highly central nodes may be sought out by e.g. marketers searching for "connectors" to trigger a "tipping point" [114]. The validity of the Tipping Point theory is not assumed here; what matters is that attackers influenced by it may attempt to discover the centrality of nodes in the social graph.

I employed the *betweenness centrality* metric in equation (3.1), as defined by Freeman [108] and calculated using Brandes' linear-time algorithm [62]. Here, σ_{st} is the number of shortest paths (*geodesics*) from vertex s to t and $\sigma_{st}(v)$ is the number of such paths which contain the vertex v. A vertex's centrality can be viewed as a "potential [...] for control of information" [108]; it is the probability that a given vertex will lie on a random geodesic.

$$C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}} . \tag{3.1}$$

Betweenness centrality is the probability that a given node in a network lies

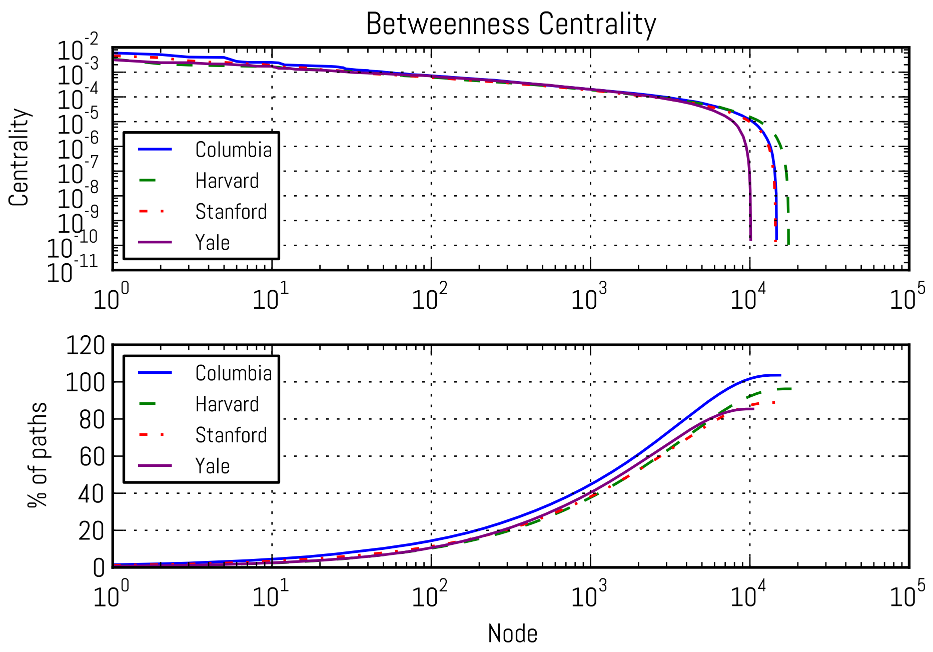


Figure 3.5: Betweenness centrality in four university networks.

on the shortest path from one randomly-selected node to another. Thus, a node with high betweenness centrality is one that can observe or influence many communication paths through the network. These communication paths are important, not because the OSN uses them to route traffic, but because real-life social networks can use them to route information via gossip. A highly central node is able to influence much of the information flowing through a social network in this way.

I used betweenness centrality to measure the effectiveness with which an attacker might influence information moving through a social network. Suppose an attacker can compromise N nodes in a social graph and use them to influence information that passes through them. If the attacker can choose which N nodes to compromise, the optimal attack strategy is to choose the most central nodes in the network, yielding the highest probability of being able to influence any communication path. If the attacker has imperfect centrality information, the probability of successful attack will be reduced. The question is, how successful can an attacker be with incomplete centrality information?

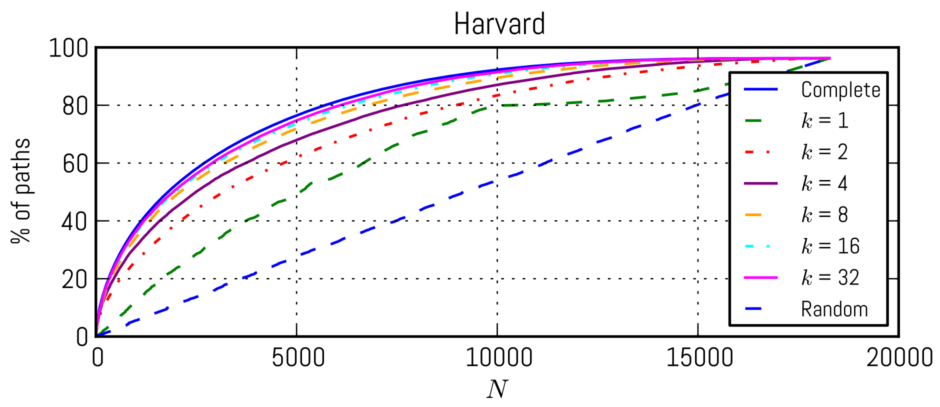


Figure 3.6: Attacker influence over messages in the Harvard network.

Figure 3.5 on the facing page shows values of betweenness centrality for the four target Facebook networks. In all four networks, the shape of the centrality curve is similar: a shallow decline in centrality across the ~5,000 most central users in each network, followed by a sharper decline across the remaining, least central users. The bottom graph shows the percentage of shortest paths in the network that involve any of the first N nodes by decreasing centrality.

To simulate the attack, I sampled two of the sub-network data sets using several values for the sampling parameter k. For each value of k, the attacker compromised N nodes according to their centrality in the sampled graph. That is, the attacker looked at a limited view of the network and chose to compromise the N nodes that appeared to be the most central.

Figure 3.6 shows the influence which the attacker would have in this scenario over messages routed over shortest paths via social links. A random result is also provided for comparison: this simulates the attacker selecting nodes to compromise without any centrality information ($k = 0$).

If an attacker randomly chooses N nodes to compromise, the number of shortest-path message routes that can be influenced increases linearly with N. If any centrality information is provided, however, his influence grows super-linearly. In the $k = 8$ case, the attacker needs to compromise $\frac{2{,}100}{18{,}273} = 11.5\%$ of the network in order to influence 50% of the shortest-path message routes through them. Compromising $\frac{5{,}596}{18{,}273} = 30.6\%$ of the network allows 75% of all

Table 3.3: Node compromises required to influence socially-routed messages.

k	N required for % route influence						
	10%	25%	50%	75%	90%	95%	99%
∞	96	466	1,794	4,736	8,788	12,419	
32	103	494	1,919	5,048	9,217	12,796	
16	107	514	1,980	5,226	9,572	13,193	
8	**113**	**559**	**2,100**	**5,596**	**10,210**	**13,916**	
4	126	660	2,492	6,440	11,355	14,859	
2	186	921	3,175	7,605	12,774	16,419	
1	823	2,137	5,089	8,951	16,668	17,994	

such routes to be influenced. These values of N are just 16–17% higher than if the attacker had full centrality information.

3.2.5 RELATED WORK

Nagaraja has studied the problem of evaluating community detection algorithms when given edges from a small subset of nodes [186]. This is related to the work above, which studies some edges sampled from all nodes, but is concerned with a different model: evolving covert networks under continuous "decapitation" attacks.

In a somewhat–spuriously-titled 2011 paper, Zhan describes how it is possible to compute some properties of a social graph properties under homomorphic encryption [245]. That work was concerned with the ability to compute such values in a privacy-preserving way; in contrast, this work is about demonstrating just how much information about the graph leaks under apparently-controlled disclosure.

3.2.6 SUMMARY

Using data from university sub-networks within Facebook, I have demonstrated that an attacker viewing a publicly-sampled social graph is able to closely approximate key characteristics of the true, unsampled social graph.

These characteristics include the *length of shortest-path routes* through the social graph: for sampling parameter $k = 8$ (as in Facebook's original public search listings), an excellent approximation for true route length can be found by simply subtracting 1 from the apparent route length.

The other characteristic I approximated is *betweenness centrality*, a measure of how connected a node is within a graph. I have shown that an attacker seeking influence over messages traversing a social graph is scarcely worse off when viewing publicly-sampled information than if he were to learn the complete social graph. When $k = 8$, this attacker needs to choose 16–19% more nodes to compromise than if he had access to the complete social graph.

This demonstrates that the decision by an OSN operator to reveal a limited subset of information — just eight friends for each user — may in fact reveal more information than was intended. This is part of the motivation for exploring alternative architectures for online social networking.

3.3 Summary of contributions

In this chapter, I presented qualitative and quantitative analyses of the privacy practices of today's online social networks (OSNs), showing how they have revealed private information about users and the social graph beyond users' expressed sharing intentions.

The qualitative analysis demonstrates three ways in which the behaviour of OSNs does not conform to the expectations and wishes of their users. First, I detailed a history of changing privacy settings in Facebook, which is currently the world's most popular OSN. These changes mean that a user who configured their Facebook profile in 2006 with very restrictive settings would today have a very *public* profile, unless active steps were taken to counter automatic changes. Second, I surveyed literature on advertiser access to private user data, demonstrating that this access has been less free than some reporters claim but less restrictive than OSNs claim. Third, I demonstrated how applications can use Facebook's APIs to retrieve information that users themselves cannot view.

My quantitative analysis centres on *social graph privacy*. I argue that the privacy of users' profile information requires the social graph also be kept private. I show how attackers can use sampled graphs provided by OSNs as public search listings to approximate important properties of the complete graph.

OSNs today have a demonstrated history of sharing private user information more widely than users' expressed preferences would dictate. The ability for OSNs to unilaterally share private user data means that users must trust their OSNs (as in the definition of "Trust" on page 15). In the rest of this dissertation, I describe an alternative social networking system called Footlights that puts users in control of information sharing, addressing problem "Explicit expressions of user intent" on page 16. I begin by describing a sharable storage service built on untrusted infrastructure that provides a foundation for a distributed filesystem.

4

Sharable storage

That's all you need in life, a little place for your stuff.

George Carlin

Current online social networks (OSNs) may not have good confidentiality properties — as shown in Chapter 3 — but they do provide their users with some integrity and availability guarantees. If Alice authenticates to an OSN using a combination of passwords, geography and social context [142, 240], the OSN can make integrity assertions to Bob such as, "Alice uploaded this exact photo". For availability, today's OSNs deliver static content such as photos through globally-available Content Delivery Networks (CDNs) [20]. Any system that improves on the confidentiality properties of today's OSNs by replacing them must also compete against their integrity and availability properties.

In Section 1.2, I proposed a distributed social application platform called Footlights to run general-purpose social applications without privileged access to user information. That platform is built on top of a global distributed storage system for private content. This storage system, together with the confidentiality and integrity properties it provides, is the topic of this chapter.

The threat model for the system is based on the behaviour of today's OSNs, described in Chapter 3. I assume that all centralised infrastructure behaves according to the definition of "Honest but curious" on page 16. That is, centralised infrastructure is relied on to faithfully perform its observable duties (such as storing content) while attempting to learn "more than [its] due share of knowledge" [116]. Specifically, the only computer that the Footlights system trusts to perform access control is the user's computer.

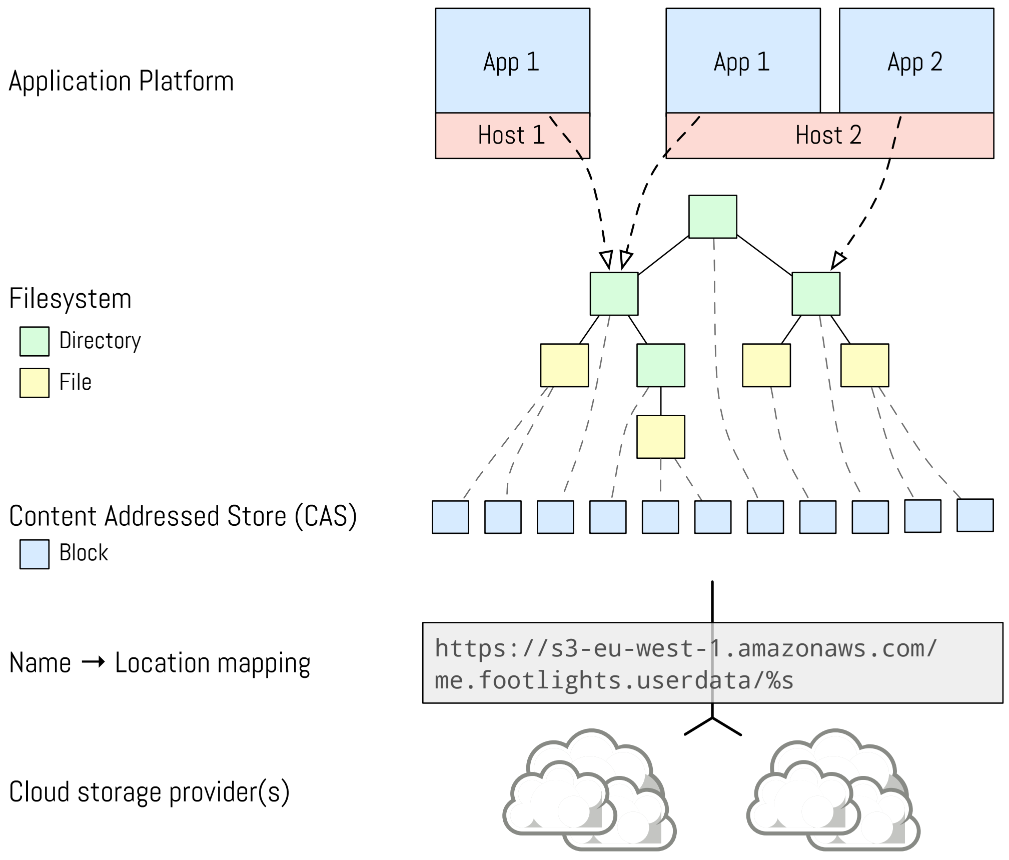

Application Platform
App 1
Host 1
App 1
App 2
Host 2
Filesystem
Directory
File
Content Addressed Store (CAS)
Block
Name → Location mapping
https://s3-eu-west-1.amazonaws.com/
me.footlights.userdata/%s
Cloud storage provider(s)

Figure 4.1: Footlights and its static storage layers.

Instead of relying on centralised OSN servers to perform access control, the Footlights storage substrate uses cryptography to achieve its confidentiality and integrity goals. Users make local access control decisions — which users and applications to share information with — that are projected into a global, untrusted storage network (Section 4.1). The substrate provides availability properties via commodity CDNs: its storage and data transfer capabilities are bounded only by the constraints of providers such as Amazon and Rackspace.

The Footlights storage system, depicted in Figure 4.1, works by storing files as small blocks of ciphertext in a content-addressed store (Section 4.2). This store is based on commodity storage purchased from an untrusted provider. Any participant in the system may validate the integrity of any block, but inspection of the relationships among blocks is only possible through explicit sharing of decryption keys. These keys are not shared with the storage provider; the provider cannot read user information and has no semantics of user identity.

The basic structure of data is a graph of encrypted, immutable blocks (Section 4.3). The entire graph is private by default, but snapshots of arbitrary subgraphs can be trivially shared with others using a constant-size communication of a name and a key, possibly via an indirect, mutable name (Section 4.4). Unlike some social sharing schemes, the decision to share content with another user implies no set-up cost or hidden changes to access control rules. Also, the system's users cannot observe how much content has *not* been shared with them, except in the most trivial sense that anyone may know the total volume of content stored by all users in the global network.

These primitives are used as the basis for a distributed filesystem (Section 4.5). Unlike centralised network filesystems, the Footlights filesystem cannot rely on garbage collection by the storage provider: filesystem semantics are not exposed to the provider. Instead, users purchase the right to store content blocks for a period of time. This is only marginally more expensive than the paid-with-privacy model of today's centralised OSNs: the cost of storing a realistic amount of user content with a current commercial storage provider is on the order of 1 USD/year (Section 4.6).

A key design goal of the system is to be compatible with privacy and performance optimisations at the network layer (Section 4.7). Users can choose their own privacy–performance trade-offs. Some will choose to access the cloud storage service via Privacy-Enhancing Technologies (PETs) in order to hide their identity from service providers. Others, more interested in low-latency, high-bitrate communication than anonymity, will find that distributed caches and CDNs allow the system's performance to compete with centralised OSNs.

For better or for worse, this network agnosticism means that the system is also compatible with traffic analysis. If users require anonymity properties in addition to privacy, the system also provides cover traffic for a communications channel that is *perfectly unobservable* (Section 4.8).

4.1 ROUTE OF TRUST TO ROOT OF TRUST

Wise men put their trust in ideas and not in circumstances.

Forbes Magazine, Vol 143, 1989 [267]
misattributed to Ralph Waldo Emerson, 1803–1882 [97]

In today's centralised OSNs, confidentiality and integrity properties are claimed on the evidence of user authentication. The provision of these properties relies on users accessing the OSN through an interface that can restrict access according to social access control lists. As a performance optimisation, authentication and access control decisions may be cached with tokens such as cookies. When these decisions are translated into the policy languages of performance-enhancing content delivery services, however, their substance can be lost.

In 2009, Joseph Bonneau, Andrew Lewis, Frank Stajano and I showed that Facebook's CDN failed to correctly revoke access to photos that were deleted by their owners [20]. The access control change requested by a photo's owner was effective when users browsed the Facebook website, but not when they visited previously-obtained CDN URLs that cached the result of an earlier access control decision. This discrepancy persisted through February 2012 [273].

The problem is that an access control decision was made in a centralised context — Facebook's data centres — but not translated into the language of the distributed CDN serving the content. Facebook could check access control rules when the user visited the OSN website itself, but the CDN could not. Not deleting old photos is a very obvious failure mode, but more subtle changes, such as modifying an access control list, can also be lost in translation between the centralised and distributed services. OSN access control is coupled to a centralised context, with access to social graphs and ACLs, making it difficult to distribute enforcement to services that do not have this access. The correctness of enforcement thus depends on the *route* by which a user reaches the protected object: whether or not a request goes through servers with access to the social information required to make the correct access control decision.

In order to improve the enforcement of confidentiality properties, many have proposed revisiting the current mix of centralised and decentralised system components. Some, like the Diaspora project [276], are partly decentralised: they break one centralised OSN into several federated ones. Others such as Safebook [79] are fully decentralised: they eschew centrally-maintained storage and instead distribute user content through peer-to-peer networks. This class of systems is different from centralised OSNs in many ways, but still exhibits the property identified above: the effectiveness of access control depends on the route by which content is reached. Diaspora's access control operates across a Web interface that is conceptually the same as centralised OSNs'. Safebook claims some of its confidentiality properties by virtue of its routing protocol, which forces accesses to go through several layers of a "matryoska" surrounding a user's data (see Section 2.3).

In these systems, the route-dependence property does not cause a breakdown in confidentiality, since users can only reach content via routes that allow access control to be properly enforced. Instead, route-dependence causes a breakdown in availability. For instance, one paper about Safebook explores the trade-offs between network churn and matryoska size, with a goal of maintaining a 90% probability that a path through the matryoska will exist [81]. This path may not have low latency or be capable of carrying large amounts of traffic: it is *a* path, and there is a 10% chance that it will not exist at all. This availability raises serious doubts about the ability of fully-decentralised services to compete with professionally-managed, centralised ones. OSNs tend not to release uptime statistics, but for comparison, Amazon, Google and Microsoft start refunding customers if their hosted application availability dips below 99.9% [318, 321, 327].

In both centralised and decentralised cases, privacy and performance seem to be at war. Facebook's access control (such as it is) breaks down when it uses CDNs to improve performance. Safebook claims to improve privacy, but does not provide the availability required of a serious Facebook competitor. I claim, however, that is it possible to have both privacy *and* performance in an OSN, as long as the system treats privacy as an end-to-end requirement [210] rather than a property of the route by which users reach content. While the availability of content might be a property of the network's core (or "the cloud"), confidentiality and integrity must be guaranteed by the edges, from end-node to end-node.

Footlights uses cryptography to project access control decisions made on local end-nodes into a global storage substrate. Information is stored on a shared cloud infrastructure, paid for as described in Section 4.6. This storage and delivery platform provides the availability Footlights requires. Confidentiality and integrity are rooted in a private key which represents a user's identity, rather than the route by which a user reaches the storage provider.

Private keys can be bound to real-world people in order to prevent the impersonation and social engineering possible in centralised OSNs or the Sybil attacks possible in anonymous peer-to-peer networks[1]. If it is important to know that "the user I am speaking with" corresponds to "the person in the next office", keys can be bound to identities via a real-life exchange of key fingerprints or CryptoIDs [197] or over channels that are authentic — but not necessarily confidential — with mutual authentication protocols such as Hoepman's ϕKE [132] or Wong and Stajano's MANA-III variant [234].

Assuming users can authenticate private keys using one of these schemes and that keys can be revoked if necessary, as discussed in Section 6.5, client software can perform end-to-end cryptography to assure users of confidentiality and integrity properties. This allows Footlights to decouple authentication from content storage and delivery. Footlights can take full advantage of professionally-managed commodity storage providers without trusting those providers to perform access control. This allows the best of both worlds: the privacy properties of a completely distributed system (end-to-end cryptography) with the availability of a completely centralised system (the storage platform).

The price of this new trade-off is *rigidity*. Centralised OSNs, which use rolling software updates and hide internal representations, can change storage formats at any time. Footlights interoperability, however, requires that formats be standardised. As shown below, this need not be complex or formal: two

[1]Douceur's Sybil attack refers to an attacker overwhelming a network with a flood of apparently-independent nodes under his control [93]. Yu et al.'s SybilGuard and SybilLimit [243, 244] as well as Mislove et al.'s Ostra [180] defend against this attack by exposing social graph information. This approach is not suitable for a privacy-enabling system such as Footlights.

binary storage formats must be specified, which could be done by a single RFC-like document; the Footlights storage system is more a protocol than a product.

4.2 CONTENT-ADDRESSED STORE

I cannot say the crow is white,
But needes must call a spade a spade.

Humfrey Gifford, "A delectable Dreame", 1580 [112]

The Footlights storage substrate holds data on behalf of many users, and the system must prevent users from modifying or removing each others' content. It is undesirable for the substrate to maintain an Access Control List (ACL) for every file, as in distributed filesystems such as OceanStore [49] (discussed in Section 4.9). Privacy is a primary goal for the Footlights storage system, so the centralised part of the system should not be trusted to check credentials or maintain content–owner/content–reader mappings. Instead, users should be able to upload content without revealing any identity information. The system may require authorisation (e.g. a token representing a micro-payment, as in Section 4.6), but it must not require authentication.

If the system were to partition user data into per-user namespaces, it would have to identify users in some way. Even if users' globally-meaningful names are not supplied, some identifier must be used to disambiguate Alice's `/photos/albums/Christmas` from a different object that Bob names the same way. This identifier might be a public key, as in self-certifying path names [170], but even that identification is unacceptable: it creates an identifier–content mapping that could be used to track the activity of individuals over time. This tracking cannot be mediated by network-level PETs, as required by problem "Linkability and anonymity" on page 21, because it is end-to-end: the server's behaviour depends on the user doing the uploading, not the communications channel being used.

If the system is not to partition the content namespace into user-specific namespaces, then all content must be placed in a global namespace that affords no special (e.g. short and human-readable) names to any user. Users should not be identifiable via the pattern of names that they use. Users should also be able to create batches of object-creation operations: Alice might create a photo album on the train, where she does not have Wi-Fi access, and let it automatically upload when she gets home. These batch operations, by definition, should not require multiple round trips to the server in order to ask, "what will this file be called?". Neither should Alice be required to pre-reserve part of the namespace: that would provide an opportunity for the system to track her via her use of the reserved names. Instead, Footlights mediates access to its globally-shared namespace by using a Content-Addressed Store.

A Content-Addressed Store (CAS) names files according to their content, rather than storage location. In Footlights' CAS, files are broken into immutable blocks, each named by a cryptographic hash of its contents.

4.3 IMMUTABLE BLOCKS

The Footlights CAS is based on encrypted, deterministically-named, immutable blocks. It is oblivious to the semantics of user files. Files are meaningful to applications, but there is no need for low-level storage mechanisms to be aware of them. Instead, the CAS works on the basis of fixed-size blocks[2], out of which files may be assembled. This distinction has previously been recognised by the CASPER system [222], a distributed filesystem that represents a file as a "recipe": a list of blocks that are stored on a backing server but may also be cached in a local CAS. This is also the design philosophy behind traditional filesystems: a filesystem is, in essence, a mapping from application-meaningful names onto hardware-meaningful disk blocks via system-meaningful objects such as inodes [173].

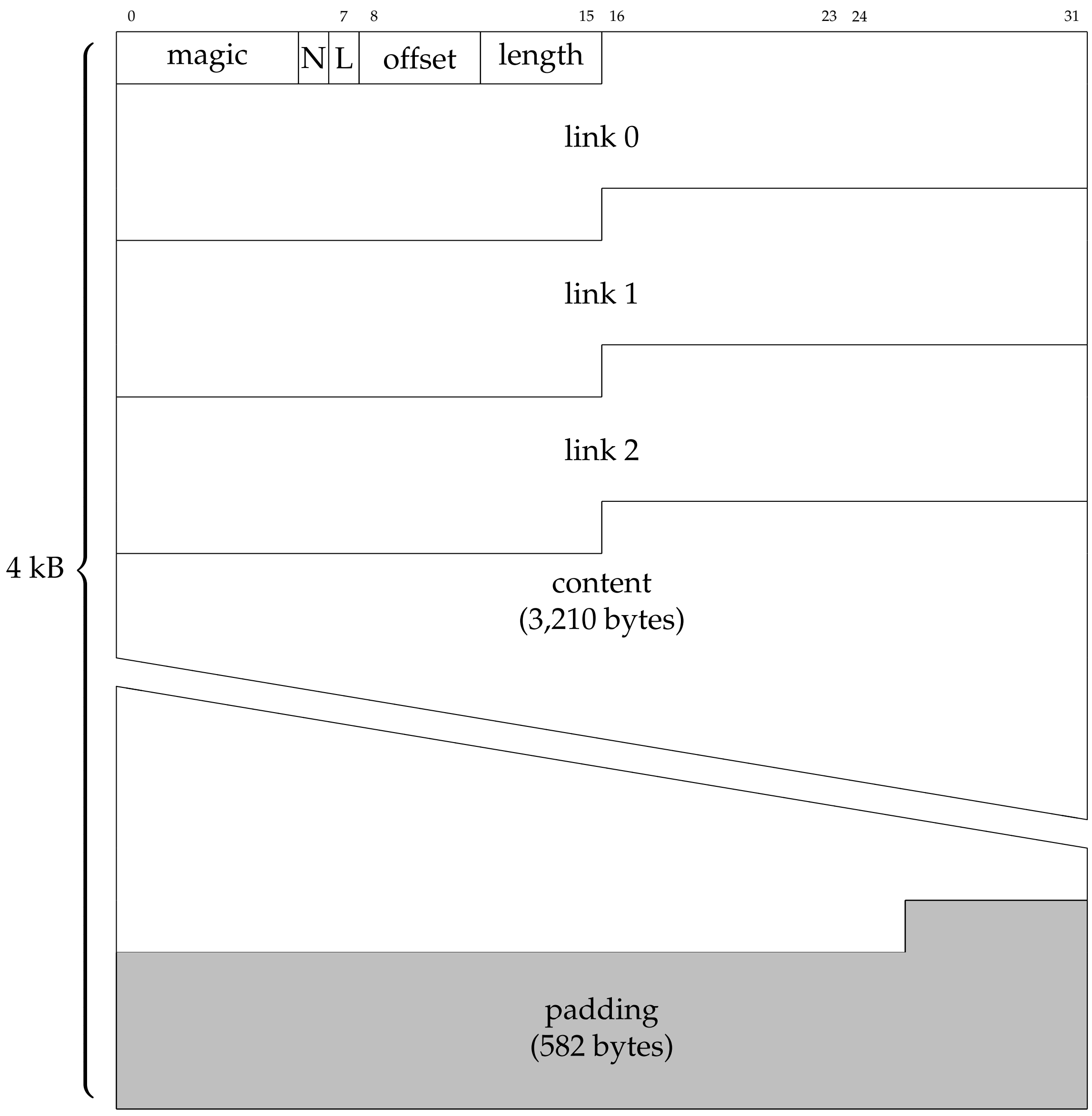
[2]The block format (Figure 4.2 on page 68) allows block size to be any power of two, but Footlights currently only uses 4 kiB blocks.

4.3.1 NAMING

Blocks in this store are named by a cryptographic hash of their content. If the hash in question admits no collisions, i.e. it is infeasible to find two blocks that hash to the same value [202], this technique provides a flat, global block namespace with deterministic names. Block names are Uniform Resource Names (URNs) [260], a special case of the generic Uniform Resource Identifier (URI) [255] that explicitly describes the content rather than location or metadata. The integrity of a block can be checked without decrypting it: if the block hashes to the name used to retrieve it, it is the correct block.

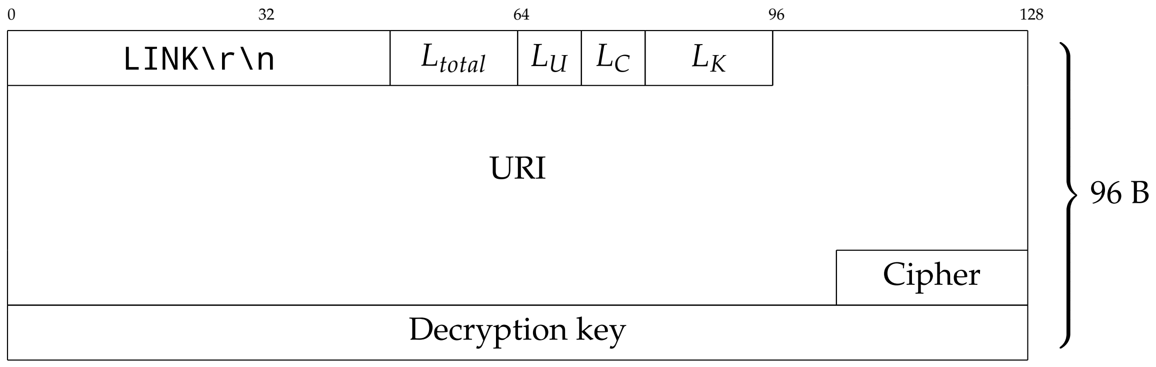
Footlights blocks named in this way are immutable; this helps solve the consistency problem that generally afflicts distributed filesystems. A filesystem that allows disconnected operation [143] such as Coda must explicitly resolve conflicts created when clients perform inconsistent writes to a shared namespace [149, 150]. Even the apparently-synchronous NFSv3 actually has weak cache consistency, so several clients can have inconsistent versions of the same file [195]. In Footlights, however, if the content of a block changes, its name must also change; it is a different block. The old block can continue to exist as a snapshot (subject to the payment model of the service), so data — such as a certain snapshot of Alice's profile — will always be consistent. The problem becomes: how can users and applications maintain consistent metadata? Which snapshot of Alice's profile is the current one? The consistency problem is reduced to the *version control problem*; this is discussed in Section 4.4.

Immutability simplifies the design and implementation of the Footlights block store because it allows sharing without co-modification. A particular block URN will always refer to the same content. This is verifiable because the name–content mapping is easily checked by any participant, so blocks can be delivered via CDN or Tor and cached anywhere in the network. The cost of this simplicity is the *garbage collection problem*, discussed in Section 4.5.



N	Total length of block: $N = \log_2 l$
L	Number of links.
offset	Offset of user content within block [B].
length	Length of user content [B].

Figure 4.2: A block with three links and 3,210 B of user data.



L_{total}	Total length of link, including magic [B].
L_U	URI length [B].
L_C	Cipher name length [B].
L_K	Decryption key length [B].

Figure 4.3: Binary representation of a CAS link.

4.3.2 STRUCTURE

A Footlights block, shown in Figure 4.2 on the facing page, contains three things: links, content and padding. A link is a pointer to another block. It contains a name and a decryption key and is represented with the binary format shown in Figure 4.3. Links provide a mechanism for structuring data into directed acyclic graphs such as hierarchical filesystem trees. Links are agnostic to the overall structure of the graph with one important exception: it is computationally infeasible to construct a cyclic graph. The name of the target block is the hash of its ciphertext, so the ciphertext must be known in order to name it. For block A to link to block B, the contents of B must be fixed before A is constructed, including all of B's outgoing links. To construct a circular graph would require finding a pre-image for a cryptographic hash; this is assumed to be infeasible for cryptographic hash functions [202].

This sharing mechanism makes it possible for users to share a snapshot of large quantities of data without any synchronisation costs beyond that of constructing the blocks in the first place. The cost of sharing is linear in the number of "things" (i.e. graphs of immutable blocks) shared, be they photos, albums or entire profiles; it is constant with respect to the size of the shared content. This property reduces the public-key cryptography burden involved in integrity checking: the graph of blocks and links is a Merkle tree [175], so signing one block protects the integrity of an arbitrary quantity of shared content[3].

The second component of a block is user content. Since files are broken into small blocks, slices of the file can be downloaded independently in parallel as in BitTorrent [77] or the earlier Jigdo (Jigsaw Download) tool [340].

The final component of a block is padding. The total size of a block in bytes must be exactly a power of two: the length of a block is $l = 2^N$ (or `l = 1 << N`), where N is a byte in the previously-shown binary block structure. Most files will not fill an integer number of such blocks precisely, leaving $2^N - \left(S \mod 2^N\right)$ byte unfilled, where S is the file size. Filling these bytes with random padding provides an opportunity for covert communication, and is an important part of the Footlights design. As described in problem "Multi-faceted identity" on page 17, users should be able to portray different facets of their online identity to different people. By hiding encrypted information in padding bits — ciphertext which is indistinguishable from its random surroundings — users can present different information to different people via the same profile root. The use of these covert bits is described in Section 4.8.

4.3.3 ENCRYPTION

Footlights blocks are encrypted with Douceur et al.'s *convergent encryption* construction [95], in which a cryptographic hash of the plaintext is used to generate a symmetric encryption key. All users who create the same block of plaintext will generate the same encryption key, yielding the same ciphertext. If the Footlights CAS is used for backing up similar files, this de-duplication effect will result in users needing to upload less information. Since the plaintext of a block includes links, no keychain is required to store the decryption keys of linked blocks.

Intuitively, convergent encryption cannot provide standard security guarantees such as indistinguishability under chosen plaintext attack (IND-CPA) [41]: when the attacker knows the encryption key, IND-CPA requires randomisation of encryption. This lack of indistinguishability is shared with Bellare et al.'s

[3]Strictly speaking, it is a directed acyclic graph (DAG) rather than a tree, but the simplicity and performance of the Merkle tree apply in all but the most pathological cases.

deterministic encryption [39], a public-key construction in which an encryption algorithm's randomising coins are generated from the plaintext itself. Bellare et al. proved that deterministic encryption provides a notion of privacy when the input plaintext has high min-entropy and does not depend on the public key. In deterministic encryption, however, there is a relationship between the encryption key and the plaintext: one is the cryptographic hash of the other.

The only formal treatment of convergent encryption in the literature is in Douceur et al.'s extended technical report version of their work [94]. In this report, a proof is offered in the random oracle model that an attacker cannot output the plaintext of a given ciphertext with probability $\Omega\left(1/n^{\epsilon}\right)$ unless that attacker can *a priori* output the plaintext with probability $\Omega\left(1/n^{2\epsilon}\right)$. This proof relies on non-standard assumptions and nothing is proven in the standard model, but such analysis is left for future work. For the purposes of the Footlights storage system, I assume that a practical attacker is unable to reverse a convergent encryption without brute-force guessing of the plaintext. Furthermore, I assume that plaintext collisions — if they exist in the hash function used to generate the key — will not result in ciphertext collisions: even if the same key is used, different plaintext will produce different ciphertext.

Since links are embedded within plaintext blocks, storage servers cannot see the structure of the block graph. The structure might be inferred by the provider via traffic analysis, however: a server might notice that whenever users download block A, they quickly come back for blocks B and C. This traffic analysis might be used to improve the quality of the provider's service through preemptive cacheing for data locality [211], but users who consider it to be an unacceptable privacy compromise are able to confuse it by constructing pathological graphs such as linked lists encrypted using an all-or-nothing strategy like Rivest's package transform [208]. This is discussed further in Section 4.7.

In 2008, Wilcox–O'Hearn — an author of the Tahoe filesystem [232] — showed that convergent encryption was vulnerable to a "Learn Partial Information" attack [358]. That is, if an adversary knows all but n bits of a file encrypted using convergent encryption, that adversary can encrypt 2^n possible plaintexts and compare the results against the ciphertext being targeted. If one of them matches, the 2^n bits of formerly-secret information are revealed. In Footlights, however, blocks are padded with random bytes. In a large file — e.g. a photo or video — the de-duplication provided by convergent encryption is highly desirable, as it reduces the amount of storage that the user must pay for. Smaller files — e.g. private messages — can contain large quantities of random padding. Implementations of the Footlights client software should ensure that small files are not left with a small amount of padding (shorter than a symmetric key of desired security). Such content should be split over two properly-padded blocks.

Storing small files within fixed-size blocks will cause an overhead cost, as discussed in Section 4.6, but this cost can be amortised across several small files if they are stored together within a block as long as they are meant to be distributed to the same recipients.

The Footlights encryption model of immutable ciphertext does not lend itself to revocation: once a user has learned a block's key, that block can always be re-downloaded and re-decrypted. I have argued elsewhere that this is consistent with real-life social sharing, and thus OSNs should not make the false promise of being able to revoke access to shared content [3]. Instead, Footlights can support Fu's "lazy revocation" model: rather than promising to immediately revoke access to a file, Footlights can simply not give access to the next file or version of the file [109]. Again, this is consistent with a social sharing model: I cannot make my friend forget a secret that I have already told him, but I can avoid sharing secrets with him in the future.

4.4 Mutable names

A complete commitment to immutability is
a commitment to never building anything real.

Yaron Minsky, "OCaml for the masses", 2011 [179]

Notwithstanding the benefits of immutable storage, applications will always require some volatile objects with stable names. Footlights provides such names through cacheable, forwardable JSON objects backed by standard Web servers.

A Footlights user's shared information can be represented as a consistent, static graph of content-addressed data blocks. This information is a snapshot, however, leading to the question: how can other users find the most up-to-date snapshot? This is the *version control problem* alluded to in Section 4.3. If Alice tells Bob in January that her personal information is in a tree whose root is x, Bob cannot be confident in February that x is still the root of her information; Alice's profile is very likely to have diverged from this immutable snapshot to a new value x'. In order for Bob to continue receiving updates from Alice, more communication is required whenever Alice updates her profile.

If Alice had a secure out-of-band channel over which she could communicate x' to Bob, the version control problem would be solved. However, reliable one-way communication channels such as e-mail use the same model of centralised access control as centralised OSNs; such a solution will not address problem "Untrusted infrastructure" on page 18.

Rather than depending on the security of external mechanisms, Footlights defines an end-to-end mechanism for translating stable names into mutable references to immutable data. Users may access this service via caches or proxies, choosing a trade-off among latency, freshness and privacy that suits their needs. Security guarantees are provided by cryptography rather than the trustworthiness of secondary channels.

In Footlights, URLs [256] can be used as mutable names. Such a URL names a mutable document that maps to an immutable block's URN (Section 4.3). The URL is a canonical, stable representation of an object such as the root of Alice's profile or the current version of a software library. The URL does not change whenever Alice updates her profile or the library vendor produces a new version, but the content that the URL references does. The example shown in Listing 4.1 on the following page is a JavaScript Object Notation (JSON) object that names an immutable block, a block-specific symmetric decryption key and a signature block.

This JSON object can be stored on any service for which URLs are well-defined, such as HTTP or FTP. This scheme is designed to be compatible with

Listing 4.1: A mutable pointer to immutable data (Base32-encoded [259]).

```
{
  "name":        "Alice's Profile",
  "fingerprint": "urn:sha-1:XH2ZD25QPHXDARTSANUHT7VF2FWIK52S",
  "key":         "AES:Z237XZFGLPANNQVRUSLHYVEL4I======",
  "signature":   "urn:sha-1:EGWBEOS2CBF7C4S7LB5ZESW7VYUMZVWN"
}
```

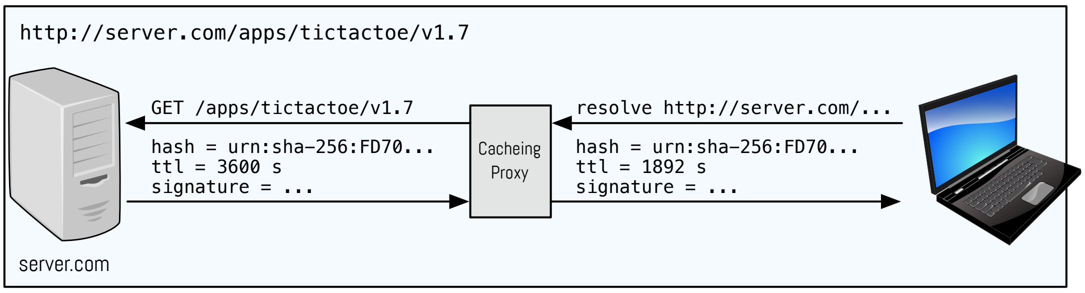

http://server.com/apps/tictactoe/v1.7
GET /apps/tictactoe/v1.7
hash = urn:sha-256:FD70...
ttl = 3600 s
signature = ...
Cacheing Proxy
resolve http://server.com/...
hash = urn:sha-256:FD70...
ttl = 1892 s
signature = ...
server.com

Figure 4.4: Resolving a canonical name.

widely-deployed cacheing systems to improve performance, privacy or both. As illustrated in Figure 4.4, name resolution does not require direct communication with the host serving the name resolution information. Caches can remember a URL→JSON mapping and serve it directly to clients. Cacheing might be used to improve performance, eliding long-distance round-trip communication with local cache hits, or to improve privacy: name resolution requests can be forwarded through anonymising proxies or peers *à la* Crowds [206].

The inclusion of a decryption key renders the linked block effectively public, but the indirection stops an honest-but-curious CAS server from reading the target without significant effort: the URL → URN mapping is not reversible. An honest-but-curious CAS provider combined with a Web spider will be able to read this plaintext; I thus apply the term *effective plaintext*, meaning blocks whose decryption keys have been made public elsewhere. Nonetheless, the CAS provider alone cannot trivially read ciphertext blocks without discovering their URL → URN mappings. URL → URN mappings could be made secret by encrypting them with standard public-key methods, but the current Footlights implementation does not support this functionality.

No security properties are guaranteed by the URL's host: nothing prevents a malicious server or cache from serving modified URNs. Rather than trust the server, Footlights only accepts URNs that it can verify by reading the signature block pointed to by the URL $\rightarrow$ URN mapping. This verification can use standard public-key signature algorithms. For instance, when viewing Alice's profile, Bob will only follow those URL $\rightarrow$ URN mappings which have been signed by Alice's signing key.

The use of caches might provide a malicious party with the opportunity to attack the consistency of the storage system. According to Brewer's theorem [63, 113], a distributed system cannot maintain both perfect consistency and perfect availability in the presence of network partitions: a trade-off must be chosen. In the Footlights storage system, an attacker in control of a URL $\rightarrow$ URN cache could force a partitioning of the network, reducing consistency or availability. Footlights makes end-to-end confidentiality and integrity guarantees with cryptography, so a malicious cache cannot poison a user's profile. A cache could refuse to reply to requests — an attack on availability — but its users can find another source of URL $\rightarrow$ URN data, such as resolving the URL itself. A more subtle attack would be a *replay attack*, an attack on consistency in which the attacker serves an old mapping rather than the current one.

A URL $\rightarrow$ URN mapping cannot vouch for its own currency, but the system can achieve Li et al.'s *fork consistency* [155]. That is, for a malicious server to hide the existence of a fresh URL $\rightarrow$ URN mapping from user Alice, it must also hide all content from her that refers to the new mapping. Hiding all such content will create an Alice-specific "fork" of the system state, presenting her with a view of the system's global state that is inconsistent with other views. In order to remain internally consistent, this fork must necessarily diverge rapidly from the view seen by other users, so it is easily detected by humans ("why haven't you replied to my message?") or by software performing a consensus protocol such as Byzantine fault tolerance [68, 151].

Name resolution information will normally be publicly available: users need not trust the access control of Web services, and making data available to the entire Internet allows users to resolve names via caches or proxies. The scheme can compose with other access control mechanisms, however: a user could generate

Listing 4.2: A filename, relative to an explicit directory block.

```
urn:sha-1:XH2ZD25QPHXDARTSANUHT7VF2FWIK52S/relative/path
```

a URL which is only resolvable by e.g. members of an educational institution or users of a corporate VPN. The URL could not be cached by proxies outside of the institution, but that constraint would complement policies of the institution that the user is relying on for their access control.

4.5 FILESYSTEM

The purpose of the Footlights storage system is to provide a storage API to distributed social applications. This API must be accessible to developers who are specialists in neither privacy nor distributed systems.

In order to meet this requirement, I have built a simple filesystem on top of the Footlights block storage CAS that can map user- or application-meaningful hierarchical names to files of arbitrary length. The details of the API, which should be familiar to Java programmers, are described in Chapter 5, but this section describes how the filesystem provides a natural unit of information for storage and sharing. That unit is a hierarchical subtree of directories based at an implicit or explicit root.

4.5.1 A ROOT BY ANY OTHER NAME

Since the Footlights CAS is a globally-shared medium, there can be no universal agreement on a directory for all users: participants cannot be relied on to make the agreement and the shared medium cannot enforce it because it is unable to look inside the encrypted blocks that the filesystem is built on.

Because of this, a filesystem must be rooted in a specific directory, which is itself composed of CAS blocks. An example of such a path is shown in Listing 4.2, where a relative component (`/relative/path`) is appended to an absolute directory URN. This mechanism allows users to have individual filesystems for content, keys, application settings, etc. without interference. The root does not always need to be explicit, however: an application might use root directories implicitly, as in `"/absolute/path/to/file"`, so long as the application platform keeps track of a root directory for each application. This allows different applications to use completely virtualised filesystem namespaces even when those applications are run by the same user on the same application host.

4.5.2 CROSS-FILESYSTEM SHARING

It is of little use for applications to have private scratch spaces if they are never able to share content with applications run by other users — the purpose of an OSN is to enable social sharing. Sharing content should not necessarily imply future sharing, however: access to one version of a photo album need not imply access to future versions. Footlights should not preclude ongoing sharing, but neither should the basic primitive for inter-user and inter-application sharing imply it.

The basis of sharing in Footlights is the immutable directory snapshot. Since the Footlights CAS is based on immutable blocks, snapshots are inexpensive: when saving a new version of a file, the system simply needs to not throw away the link to the old version. This property is shared with the ZFS file system [55] and the Git revision control system [161], both of which also use hashes to name blocks of content. The cost of this approach is seen in garbage collection, discussed in §4.5.4.

Time-varying content can be shared via indirection through a mutable URL (Section 4.4) or explicit publishing of updates. This is different from filesystems that share the one-logical-root-per-application property of Footlights, such as Plan 9's `/proc` filesystem [199, 200], inspired by Killian's ProcFS [141]. In the `/proc` filesystem, as implemented by Plan 9, Solaris, the BSDs and Linux, some particulars of every process on the system are exposed via paths such as `/proc /{PID}/fd`. Delegating access to a portion of this filesystem necessarily implies ongoing access to up-to-date information. If a system like Capsicum [9] is used to delegate the `/proc/17/environ` directory to an untrusted process, that process will always be able to read the environmental variables of the process with PID 17. There is no way to delegate a snapshot of only the current environmental variables without copying. By contrast, the Footlights sharing mechanism uses immutable snapshots by default. It is possible to provide ongoing access to changing content, but the default is to share content as it currently exists.

4.5.3 WRITING TO SHARED FILESYSTEMS

Since each application instance is the master of its own filesystem, no other application has the authority to modify its contents. This provides predictable semantics to the application: no other application will concurrently modify content. The filesystem is not required to mediate conflicting writes to the distributed filesystem. Mediation would imply a burden for application developers: distributed filesystems such as Coda that support concurrent writes need application-specific conflict resolvers [149, 150].

On the other hand, the ability to modify shared content is a desirable feature. If application A shares a subset of its private filesystem with application B, then B cannot directly write into A's filesystem: it is an immutable snapshot. B can, however, modify a local version of the shared filesystem and send A an immutable snapshot of the changes that B would like to see incorporated. A can then decide whether or not to incorporate the changes and update other applications with the new version. This is similar to the development model employed by the distributed development service Github [82]: users can trivially fork an existing Git project — an immutable tree named by a cryptographic hash of its contents — and modify a local copy, then submit "merge requests" back to the authors of the original project. This allows a locally-mutable filesystem to be incorporated into a globally-immutable filesystem snapshot.

4.5.4 GARBAGE COLLECTION

The Footlights CAS cannot inspect the content of filesystem blocks, nor do users supply it with metadata such as reference counts. Together, these properties prevent the underlying storage provider from deleting stale blocks; that is, it cannot perform *garbage collection*.

Instead of performing garbage collection, the underlying provider of Footlights block storage can resort to a fee-for-service model, in which users pay the provider for de-duplicated storage of encrypted blocks for a fixed period of time. This model is plausible because the costs of storage are so low, as described in Section 4.6.

4.6 Cost

The plausibility of a privacy-preserving storage and sharing service depends on the cost of operating the service. Current OSNs pay for data storage and transport with targeted advertising. Privacy-preserving advertising schemes have been proposed [48, 75, 126, 137, 196, 224] which are technically sound, but it is currently unclear how much revenue they will be able to generate. The question is, how much revenue is required?

The Footlights storage service uses a commodity provider of storage and content distribution. I have estimated the costs of running Footlights on top of Amazon, Google App Engine or Rackspace storage, using either the storage service's content delivery network or the NetDNA CDN. The costs published by these services are given in Appendix A, "Content Delivery Networks".

Google App Engine provides both storage and distribution services on the basis of a fixed rate, minus a free monthly quota [336]. Rackspace provides a similar service, but with a lower rate and no free quota [330]. In both cases, content delivery can be performed by an external content delivery network such as NetDNA, which offers tiered rates based on total volume [337]. Amazon provides tiered rates for both storage and content delivery, but as shown below, its cost structure is unsuitable for Footlights.

Before answering the question of how much the Footlights storage service would cost to run, I first turn to the question of how much storage is required. In February 2012, the world's largest OSN — Facebook — reported that it had 845 M active users and stored 100 PB of photos and videos [333]. On average, this is just 113 MiB of photo and video content per user. I assume that these values will be similar for Footlights users: as Figure 4.5 on the next page shows, a photo management application (§5.4.4) storing photos in the filesystem incurs an overhead of 2% (80 B/block) plus a constant 48 kB, according to a least-squared regression.

To estimate the cost of running the Footlights storage system, I assume that

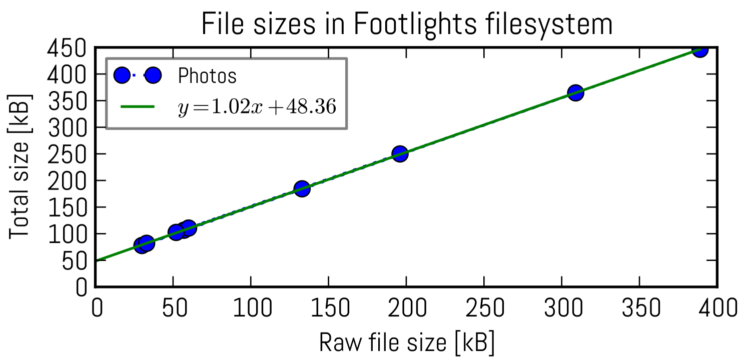


Figure 4.5: Bytes required to store photos in the Footlights filesystem.

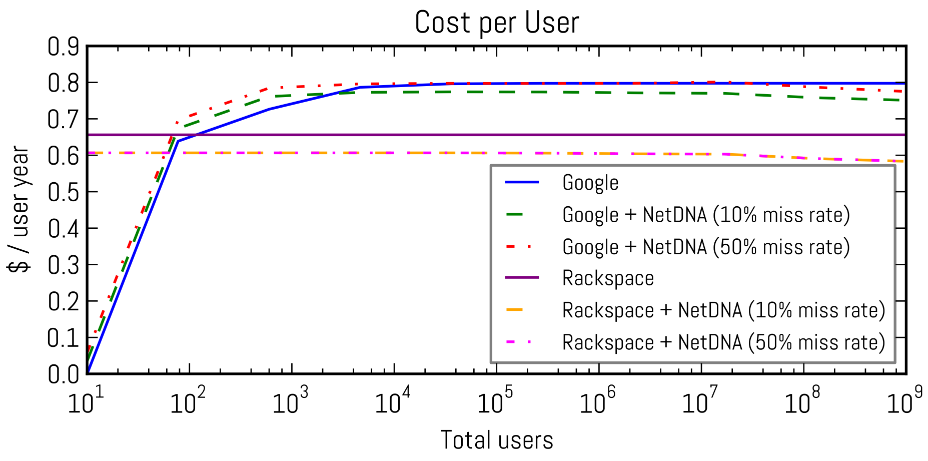


Figure 4.6: Cost of operating the Footlights storage service.

each user will store 4×113 MiB in the CAS, including de-duplicated photos and videos, other user data (e.g. text) and a margin for expansion. I also assume that each user will download one profile worth of other users' data each month.

Based on these assumptions, the cost of running the Footlights storage service in several provision scenarios is shown in Figure 4.6. When the Footlights OSN is very small, Google App Engine's free quota makes it more economical than Rackspace, but Rackspace's lower fixed costs make it more economical once the service has more than approximately 100 users. The use of a third-party CDN reduces total costs, particularly as the number of users increases

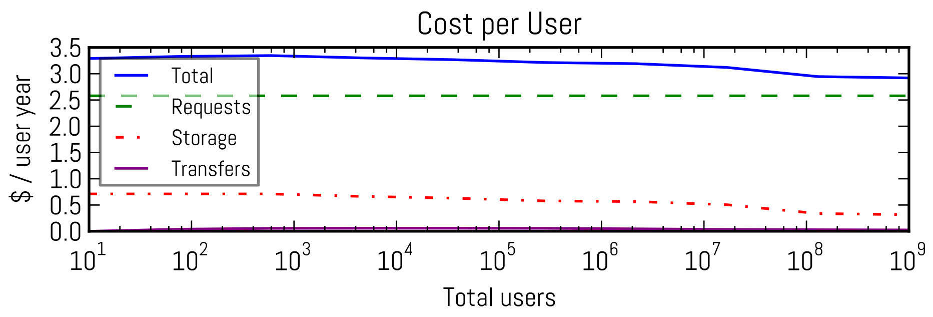


Figure 4.7: Cost of basing Footlights on Amazon Web Services.

past 10,000, although this difference is more pronounced in the Rackspace case, since Google App Engine's free quota means that the use of a CDN will increase costs when there are few users.

Not shown in Figure 4.6 on the preceding page is the cost of Footlights provision based on Amazon Web Services. Amazon charges a fixed rate for HTTP requests (per 1,000 PUT or 10,000 GET operations) [317]. Footlights splits files into 4 kB blocks, each of which requires an independent GET operation, so in the Amazon case — or the new Google Cloud Drive — the cost of HTTP GET requests dominates that of storage and data transfer. This is illustrated in Figure 4.7.

All of these costs are based on publicly-available figures, which do not provide complete information about the market. Amazon does not disclose its highest-volume tier prices publicly, and other major CDNs such as Akamai and Limelight do not disclose any prices. These undisclosed prices will be lower than the publicly-available ones, further reducing the cost of running the Footlights storage system.

Furthermore, if Footlights grew to serve millions of users, an even more efficient platform might be built to serve static content. Current commodity storage platforms allow variable-sized, mutable content. A system that mapped fixed-size, immutable, content-addressed blocks directly to physical blocks in a storage array might prove more efficient to build and operate than a general-purpose storage platform.

Even without such optimisations, however, it is feasible to provide a Footlights storage service based on existing infrastructure for less than one US dollar per user-year. Users might pay such costs directly, perhaps through a premium text message service, but it is also plausible that these costs might be borne through privacy-preserving advertising.

4.7 PRE-CACHEING AND TRAFFIC ANALYSIS

Storing user content as a sea of opaque ciphertext blocks accomplishes a confidentiality goal: it prevents the storage provider from reading content that has not been explicitly shared with it. It also reduces the amount of information that is available for performance optimisation. This information loss is part of the cost of improving online privacy, but it can be offset by clients: voluntarily in exchange for reduced latency or involuntarily through traffic analysis. Traffic analysis can be defeated, however: clients are able to trade latency for anonymity, sacrificing performance in order to prevent traffic analysis.

Scellato et al. showed that the social graph can be used by Content Distribution Networks (CDNs) to improve performance via pre-cacheing [211]: if Alice is looking at Bob's photo album, there is an increased probability that Alice's and Bob's mutual friends will look at the same album in the near future. This information can be used to pre-cache the album in locations close to these mutual friends and reduce the latency with which they can retrieve the content. This source of metadata is removed by Footlights: the storage provider cannot read metadata such as who is looking at which blocks or how they are linked. This confidentiality gain has the potential to cause performance costs.

It is clear that information about user intent can improve cacheing decisions. It is also clear that Scellato's social cascade detection is one way of inferring intent probabilistically. However, this is not the only way of obtaining information about users' intent for future data accesses. In 1995, Patterson, Gibson et al. considered the cacheing problem in an economic sense: every cache hit provides a benefit, but every full cache buffer implies an opportunity cost — that buffer cannot be used for other information [194]. This economic perspective might be extended to consider the costs of both computation and access to private information. In this view, socially-informed pre-cacheing has a privacy cost: it requires the lives of users to be laid open and bare before the CDN. Scellato et al. did not compare the costs and benefits of socially-informed pre-cacheing with other sources of information about user intent. Sources of intent information compatible with a privacy-preserving storage system include:

Explicit declaration Client software, when downloading content from a Footlights CDN, could simply state explicitly which blocks it wishes to download before it downloads them. This is analogous to the application practice of declaring I/O hints to the operating system which Patterson et al. found "always increases throughput" [194]. This explicit declaration of intent will not improve the cache hit rate for the first block that a client downloads, but it may improve the hit rate for subsequent blocks. Assuming that the CDN is able to transfer data more quickly internally than clients can download them — a reasonable assumption, as this is the *raison d'être* of a CDN — the CDN should be able to pre-cache all but one of the blocks requested by the user.

Traffic analysis CDN providers could also infer likely future access patterns through traffic analysis. CDNs could store patterns such as "when block X is accessed by any IP address, that IP immediately accesses block Y", each of which is associated with a probability. Such a strategy carries a clear algorithmic cost: the CDN must store traffic information, compute probabilities, etc. This cost is not new, however: it is simply a translation of the non-monetary cost to privacy that was previously treated as nil.

In addition to performance enhancement, traffic analysis can also be used to identify users and their social graph [34, 182, 205]. If a storage provider and CDN wished to identify Footlights users, it could record all accesses to content and identify which IP addresses access the same content, as well as which IP address uploaded that content. Such traffic analysis could easily identify cliques of stable IP addresses, but users can obfuscate their activities.

Taking lessons from the security economics literature [30], the user seeking protection should not depend on other users taking costly obfuscation actions. When downloading content, the user who wishes to frustrate traffic analysis can choose to download via systems like Crowds [206] and Tor [91] which hide the identity of the downloader. This action has a cost — using relays will slow the download — but it is borne by the user seeking to frustrate the traffic analysis. The user chooses the trade-off.

Similarly, when uploading content, a user may wish to hide its structure. The sea-of-blocks technique hides overt structure, but a malicious storage provider could perform traffic analysis to infer structure. If a directory contains two files, those files' encrypted blocks will often be requested immediately after those of the directory. Once again, obfuscation comes at a cost: a user's blocks could be organised pathologically, e.g. in a linked list rather than a tree and packaged with an all-or-nothing transform.

Rivest introduced all-or-nothing encryption as a method of slowing brute-force attacks even when using "marginal" key lengths such as DES' 56 b or the 40 b imposed by US export restrictions at the time [208]. Rivest's initial all-or-nothing encryption mode, the *package transform*, applied an inner counter-mode encryption to plaintext, the key to which was appended to the *pseudo-message* (the transformed plaintext) but XOR'ed with hashes of the preceeding pseudo-message blocks. The pseudo-message was then encrypted in the conventional way, so an attacker guessing the outer key would need to decrypte the entire message rather than a single block; this added a significant work factor to brute-force attack. Boyko later proved (in the random oracle model) that the Optimal Assymetric Encryption Padding (OAEP) algorithm constituted an all-or-nothing transform whose security could not be substantially bettered by any other such transform [60].

The application of an all-or-nothing transform to the plaintext of Footlights blocks would increase the burden of creating them — such transforms have a computational cost — but it would especially increase the cost of downloading them, as clients would need to download an entire profile before they could decrypt any part of it. As in the downloading case, however, the choice to obfuscate is made by the user who requires protection, and no client with a desire to take shortcuts is in a position to undermine it.

Users can thus choose to trade off performance for privacy, obfuscating access patterns to frustrate traffic analysis. The choice to perform obfuscation lies with the user who wants to be protected from traffic analysis. These mecha-

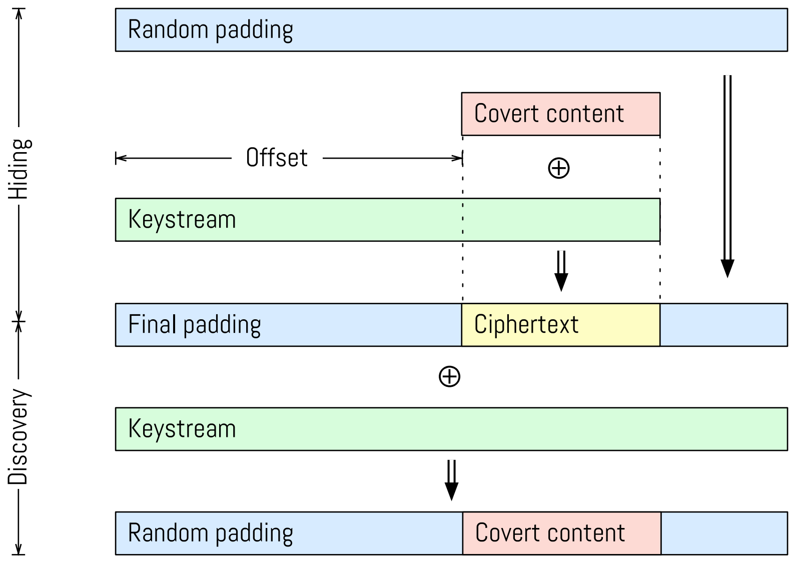


Figure 4.8: Embedding covert content in random padding.

nisms will make traffic analysis more difficult, but not impossible. In order to construct a *perfectly unobservable* communications channel, users must employ more costly techniques from Section 4.8.

4.8 Covert communication

In order to support multi-faceted identity (problem "Multi-faceted identity" on page 17), it is important that users be able to share limited content without revealing that the sharing is limited. That is, there should be no "known unknowns": Bob should not be able to determine that Alice has published a piece of content unless she has chosen to share it with him.

As described in Section 4.3, Footlights blocks are padded to a fixed length with random data. Assuming that the system is implemented with encryption primitives that can be modeled as pseudo-random permutations [40], i.e. an attacker cannot distinguish between ciphertext and random data, block padding can be used to hide covert information.

The technique is illustrated in Figure 4.8. When hiding information, the block's creator Alice chooses an arbitrary offset o within the padding. This offset allows her to hide multiple covert messages at different locations within the random padding. She then uses a stream cipher to produce a keystream of length $o + l_C$, where l_C is the length of the covert content. Finally, she XORs the covert content with `keystream[o:o+lc]` and embeds the resulting ciphertext at offset o within the padding. In order to recover the covert content, the receiver Bob uses the shared key to generate keystream of the same length as the block's padding. He then XORs these together and searches for meaningful content embedded in the random padding.

The stream cipher key may be randomly-generated, in which case Alice must communicate it covertly to Bob. Alternatively, it may be established by a Diffie-Hellman exchange [90], using parameters specified by Alice and Bob in their respective profiles.

This protocol involves no explicit message authentication code (MAC). This is unusual for a protocol that uses a stream cipher for encryption: a similar design choice in IEEE 802.11's original Wired Equivalent Privacy (WEP) led to exploitable vulnerabilities and the eventual introduction of the Michael message integrity code [235]. The protocol described here is not vulnerable to these attacks, however.

In the WEP attacks, an attacker could modify all ciphertext bits, leading to linear changes in both content bits and the linear CRC32 checksum. This allowed attackers to modify the content of messages without detection [57]. In the Footlights case, however, a block is named by a hash of its contents; if an attacker changes any ciphertext content, it will be detected because the content will not hash to the same value. Even if an attacker finds a second pre-image for the hash function, the malicious block must pass a second integrity check. After decryption, the block's entire plaintext is hashed and the result must be equal to the decryption key which was used to decrypt the ciphertext.

This technique for embedding content within a block's random padding can be used to hide covert links (§4.8.1) or content (§4.8.2) within otherwise-overt Footlights blocks.

4.8.1 LINK HIDING

Footlights can support multi-faceted identity by hiding links within random padding using the technique described above. By using different offsets and different keys, a block's creator can embed one link to be read by Alice and another to be read by Bob within the same Footlights block. This allows a user to express different profiles to different friends using the same root block, and neither Alice nor Bob can detect facets that are not shared with them.

A user may wish to hide more links than will fit in the padding of one block, but links can be hidden in any number of overt blocks, and the structure of the block graph is entirely under the control of the user's client software. A shared key that is used to hide links could also be distributed to multiple users in an implicit group, reducing the number of links that must be hidden without revealing the grouping to any user. Footlights cannot stop users from comparing notes with each other, examining the differences between what has been shared with each of them. I do not consider this to be a technical problem, however, but a social one: it is as if my friends met physically to reveal and compare everything that I tell them. No technical measure in an OSN can stop people from gossiping.

However, the differences between the blocks that Alice downloads and the blocks that Bob downloads may be visible to a malicious provider of underlying storage who performs traffic analysis. In order to hide content not just from friends but from the provider, a similar technique can be used to hide content rather than links.

4.8.2 MESSAGE FORWARDING

A sea of immutable blocks with random padding provides an opportunity to set up a *perfectly unobservable* communications channel. This channel can be used to relay covert messages through a social graph without detection.

I pessimistically assume that the underlying storage provider is always able to link uploads and downloads to a particular user, identified by a static IP address, and that this channel is the only communications channel available to users. That is, the storage provider is the *global passive adversary* [89, 205].

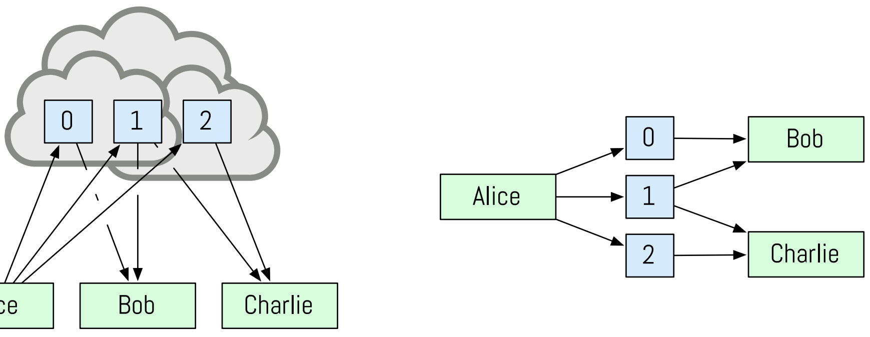


Figure 4.9: Block upload/download is equivalent to multicast messaging.

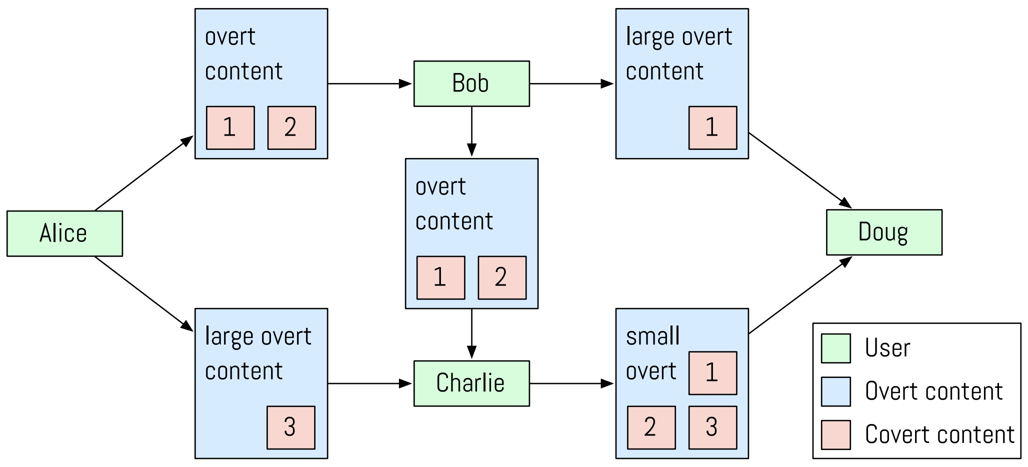


Figure 4.10: Random padding viewed as spare message-carrying capacity.

Whenever a block is uploaded by a user S and downloaded by a set of users R, the global passive adversary can link the upload and download transactions together and interpret the result as a multicast message from sender S to recipients R. The equivalence of these two views is illustrated in Figure 4.9. Even though users hide the content of messages from the global passive adversary, traffic analysis reveals the structure of communications and thus the structure of the social network.

Inside this overt communication graph, however, the random padding at the end of each block can be used as excess capacity for carrying covert messages. These messages may be hidden links as in §4.8.1, but they can also be packets to be routed around the network. If the random padding in each block is viewed as a channel from one node in the social network S to all block recipients R, a flow network as described by Ford and Fulkerson [107] can be constructed through the overt Footlights blocks, as shown in Figure 4.10 on the facing page.

If users enqueue covert messages to be embedded opportunistically in the future, then this communications channel can be *perfectly unobservable*. As long as the decision to send a message is not influenced by covert messaging requirements, the overt communications graph will be the same whether or not it carries covert packets. The only difference in the two graphs is the content of the blocks, given in equations (4.1) and (4.2).

$$B = \{C_O + R\}_{k=h(C_O+R)} \tag{4.1}$$

$$B' = \{C_O + C_C\}_{k=h(C_O+C_C)} \tag{4.2}$$

In these equations, C_O is the overt content of a block, C_C is covert content, R is random padding and h is a cryptographic hash function. The results, B and B' are the ciphertexts of a block without covert content and a block with covert content. Under the random oracle assumption[4], an adversary inspecting B or B' will gain no information about the plaintext $C_O + R$ or $C_O + C_C$. The adversary cannot distinguish between the plaintext with embedded covert content and the plaintext without it; the channel over which the covert content is communicated is *perfectly unobservable*.

This perfect unobservability is in contrast to privacy-enhancing technologies such as mixes [70, 84] and Tor [91], the use of which can be regarded as a "something to hide" signal. It is also different from steganographic systems [32, 198], in which detection is a probabilistic affair and "security level" is based on an arbitrary choice of acceptable false-positive and false-negative rates [198]. In steganographic systems, knowledge of the cover image exposes the steganographic payload; in Footlights, random block padding means that knowledge

[4]Proving indistinguishability between B and B' in the standard model may be a difficult task, since each uses a different key but both keys are related to the plaintext. Constructing such a proof is left for future work.

of overt plaintext does not imply knowledge of the resulting block's ciphertext.

Unlike traditional steganographic file systems such as StegFS [172], the Footlights covert channel is *stable*: covert content cannot be unwittingly overwritten. McDonald and Kuhn's StegFS is inspired by Anderson, Needham and Shamir's description of a theoretical steganographic file system [31]. Both systems admit to the presence of one level of encrypted content, but hide the presence of further levels. Users unlock levels ("directories") of hidden information by supplying passphrases, but can plausibly deny the existence of further levels of secrets. This plausible deniability comes at a cost. Just as it is impossible (under the assumption of block cipher indistinguishability) for an inspector to determine that further covert content exists, so it is impossible for the filesystem to avoid overwriting locked covert information. The filesystem reduces the probability of overwriting through replication, but it is still a probabilistic guarantee. This guarantee, predicated on complete control over how much content is written relative to the size of the host partition, cannot scale to a storage medium shared by millions of users and backed by an untrusted cloud provider. In contrast, Footlights uses immutable, content-addressed, encrypted data that can be verified by any party in the system, so it provides a stable shared medium in which covert messages will never be overwritten, though they may expire.

Covert messages can be routed through the network of overt messages using a routing scheme appropriate to opportunistic, one-way delay-tolerant networks (DTNs). DTNs, as named by Fall [99, 100], are *store-and-forward* networks; they route communication among their nodes without requiring that all the links in a route be available at the same time. Most DTNs are either opportunistic or one-way: opportunistic schemes assume that two high-bitrate radios will be in close proximity for some period of time, whereas one-way communication is typically used with predictable contact, e.g. space vehicles whose positions can be predicted. The perfectly unobservable channel afforded by Footlights, however, is both opportunistic and one-way. Designing a routing protocol that performs well under these constraints is left for future work.

4.9 RELATED WORK

Blaze's Cryptographic File System (CFS) [51] allowed Unix users to transparently encrypt a local filesystem mounted at `/crypt`. The resulting ciphertext could be stored on networked filesystems such as NFS and AFS, protecting user information from other users of the same distributed filesystem. Authority was concentrated in a per-directory key: sharing encrypted content between users required the sharing of keys. CFS was the first filesystem to use this abbreviation. Confusingly, both Cedar and Chord — described below — later re-used the acronym without citation, apparently unaware of Blaze's work.

Grolimund's Cryptree is a scheme for managing keys in cryptographic filesystems [124]. Cryptree uses the term "link" in a subtly different way from Footlights: in Cryptree, a link is a construction that allows one key (e.g. of a subfolder) to be calculated from another key (e.g. of a parent folder). The exact construction is not specified, but the authors cite work such as Chien and Jan's hierarchical key assignment [72]; this work relies on tamper-resistant hardware to translate keys, but a standard key derivation procedure would also work.

The Cedar filesystem [111] used immutable files to eliminate the cache coherency problem in a network file system. Cedar was primarily used to support programming, so it had a strong focus on source code. Files were named with an explicit version number, so the problem of distributed cache coherency was translated into the problem of version control, a natural problem for programmers. It seems to have been influential for distributed revision control systems such as Git [161], which uses the same model of immutable files but goes a step further with immutable snapshots of directory trees, as well. Cedar required trusted file servers, which Footlights seeks to eliminate, and it was also vulnerable to accidental misuse: users could write to CFS (Cedar File System) files via non-CFS mechanisms, violating the assumption of immutability. Footlights, in contrast, uses a naming scheme that permits content-based integrity checks.

In 1998, Anderson, Needham and Shamir described two theoretical steganographic filesystems [31]. The first relied on linear combinations of random data: a $k \times n$ matrix was constructed, where n was the size of a file or directory, and k the number of random cover files in the system. Users could store up to $m \leq k/2$

files in this matrix by XOR'ing cover files specified by a strong passphrase P, computing an XOR difference D between the result and the desired file, then XOR'ing D into the matrix of cover files. By using a $k \times k$ *extraction key* matrix, the user could build a linear hierarchy of *security levels*, each of which provided no information about the next level up. Users unlocked levels ("directories") of hidden information by supplying passphrases, but could plausibly deny the existence of further levels of secrets. This technique was vulnerable to a known-plaintext attack, however, and it had poor performance properties: to save a covert file would require writing 16–50 times more data than a normal file.

In the second concept, which inspired McDonald and Kuhn's StegFS [172], covert data was encrypted and written to disk in pseudo-random locations. Blocks of data were replicated in several locations to reduce the probability that all copies of a file might be overwritten. This system chose a different set of trade-offs: it required ~5× redundant writes rather than 16–50×, but at a cost of a lower *load factor* (space utilisation). In this system, lower security levels could not know where higher-level files might be stored, so overwriting of covert data was a probabilistic risk.

Both of these steganographic filesystems assumed absolute control of the underlying storage medium. Since any new file might be written to any bit of the $k \times n$ matrix or any location in a Linux partition, the ability to write covert content implied complete trust by the owner of the filesystem. Users could not share a steganographic filesystem. In contrast, Footlights provides a storage medium which can be shared among mutually distrusting users, in which writes performed by one user cannot degrade the security of others. Instead, one user's content will *enhance* the security of others' in all but the most pathological cases by providing cover traffic.

In 1998, Gopal and Waters thought that traditional filesystems were reaching the limits of usability, and sought to integrate "content-based access" into a hybrid filesystem which they called HAC ("Hierarchy and Content") [120]. The terminology used by this paper is very different from its use in this dissertation, however; some disambiguation may aid clarity of thought for readers who are familiar with similar work from this era. Gopal and Waters, in keeping with Gifford et al.'s Semantic File Systems concept [110], used the term "content" to refer to a semantically rich understanding of file attributes and contents. The Semantic File System (SFS) incorporated "transducers" that could parse and describe source code, mail files and New York Times articles. HAC took this concept further, allowing query "directories" to be more than read-only snapshots of query results, but it too required the system to have deep knowledge of file contents in order to be useful. As any screen-scraper knows, scouring New York Times articles to apply semantically-meaningful names can be a brittle activity. In Footlights, by contrast, the term "content-addressed" is used to refer to a very shallow semantic: the bytes in a ciphertext block. The storage substrate is prevented by cryptography from reading deeper into content, but cryptographic hashes provide a robust naming scheme. Applications are expected to interpret their own data, encouraging lock-step co-development of formats and parsers[5].

A content-addressed store, in the sense that Footlights uses, was created by FilePool NV in the late 1990s and later acquired by EMC. A FilePool patent filed in 1999 describes how files could be identified by a cryptographic hash of their contents and collated together with an ASCII representation called an "e-Clip" [67]. This patent has not been cited by the later research literature.

OceanStore [49, 207] is a model proposed as a "true data utility". It is based on the premise that "information must be divorced from location": it uses semi-trusted replica servers to store and distribute information. Service is paid for by a monthly fee, although explicit costs are not discussed. Like Footlights, users' data is stored encrypted on the utility's servers, so only users — not service providers — can read plaintext.

Also like the Footlights storage system, OceanStore names objects with cryptographic hashes. Unlike Footlights, however, OceanStore names are hashes of an owner signing key and an owner-specified name — Mazières et al.'s self-certifying path names [170, 171]. This means that users storing content in the system must identify themselves via a signing key.

This identification is taken a step further when dealing with shared, writable content. OceanStore names are mappings to mutable content; they can be mod-

[5]The authors also used the term "fingerprint" in a very different sense from later CAS work: one of the authors had an interest in biometrics.

ified if a user's signed update command is accepted by a Byzantine agreement protocol among "primary tier" OceanStore replicas. These replicas are not trusted with plaintext, so objects' Access Control Lists (ACLs) must be public. Whereas Footlights ensures that each client can choose its own security/performance trade-offs, OceanStore publishes ACL entries publicly. These entries are only for object writers and they refer to signing keys rather than identities, but Footlights eliminates this publishing requirement — and the trust placed in primary tier replicas — by using completely untrusted content-addressed storage.

The idea of self-certifying path names — though not the name — is used by Graffi et al. in their discussion of encryption within peer-to-peer–based social networks [121]. This work recognises that a system "must be compatible to [sic] common replication mechanisms and caching mechanisms" in order to provide the availability properties that users expect. In this system, users can determine how many users' public keys a file has been encrypted to; this conflicts with problem "Multi-faceted identity" on page 17.

The idea of content-addressed storage was developed by Freenet [76], which named content with "content-hash keys" plus an indirection mechanism with signed namespaces. Ratnasamy et al.'s Content Addressed Networking paper [204] further develops the content addressing idea by proposing the distributed hash table mechanism. The paper cites Freenet as an example of a file sharing system, but fails to note Freenet's content-hash keys as an example of content addressing.

The third CFS is the Cooperative File System [83]. CFS is a read-only filesystem built on top of a block store called DHash, which is itself built on the Chord distributed hash table [221]. CFS can use content addressing or certified namespaces *à la* OceanStore, above. CFS-on-Chord is conceptually similar to Footlights-on-Amazon, but not in the details: CFS does not tolerate malicious participants in the system, nor does it provide lookup anonymity. The authors state, "it is expected that anonymity, if needed, would be layered on top of the basic CFS system".

Quinlan et al.'s Venti archival system (2002) [203] used many of the same low-level techniques as the Footlights block store, including immutable blocks of data named by a hash of their contents, block de-duplication at the back end and trees of content that can be represented by a single "pointer" block. Venti provided an archival block store that could be safely shared by multiple clients without co-ordination or mutual trust and which provided implicit integrity checks. Footlights' demo file manager (§5.4.5) is functionally similar to Venti's vac: it converts files of arbitrary size into hash references of fixed length. Footlights takes the Venti work a step further: rather than providing a trusted block store which can be shared by untrusted clients, Footlights allows untrusted storage providers to supply back-end storage for mutually suspicious clients.

Muthitacharoen et al.'s Ivy [183, 184] is a writeable filesystem built on the DHash block store from the Cooperative File System (CFS) described above. Ivy is a log-structured filesystem: users append immutable entries to their own logs which reflect changes to the filesystem state. A particular user's view of the filesystem can be calculated by combining trusted logs together. Users can create a shared filesystem by generating a "view block" that certifies the shared filesystem as a defined set of user logs. The Ivy filesystem was found by its authors to be 2–3 times slower than NFS. Like Ivy, the Footlights storage system requires individual users to maintain their own filesystem namespaces but allows namespaces to be viewed collectively as a shared filesystem. Unlike Ivy, Footlights is based on current snapshots: retrieving the current state of a filesystem involves block retrieval, not log replay and block retrieval. Ivy does allow snapshot blocks to be periodically constructed, but in Footlights, it is the current snapshot which is the first-class primitive; logs of a filesystem's history are a construction based on snapshots.

Tolia et al.'s CASPER system (2003) [222] used a content-addressed store to opportunistically improve the performance of a remote file system accessed via Wide-Area Network (WAN). CASPER viewed files as "recipes" containing lists of CAS blocks. These blocks could be cached anywhere in the network, leading to latency improvements if a client did not need to contact the remote file server directly. The blocks themselves contained pure content, necessitating the "recipe" indirection. In Footlights, by contrast, less structural information is revealed to the (potentially adversarial) storage service: encrypted blocks which contain links are indistinguishable from those that do not. CASPER does not address encryption directly, although the paper does suggest that convergent encryption [95] could be used for de-duplication of CAS blocks.

The SUNDR network file system created by Li et al. [155] is an example of a shared file server that is verified by clients using public-key cryptography. SUNDR names blocks according to their SHA-1 hashes, like OceanStore, CFS and Venti. Like the Ivy filesystem, SUNDR relies on clients signing the actions they take to modify the shared filesystem. Unlike Ivy, SUNDR centralises the activity log and block store for performance and to commit to an ordering of operations: its authors claim that Ivy's distributed logs do not provide order consistency. SUNDR allows clients to interact with a fork consistency model (described in Section 4.4), but its access control model is otherwise conventional, with users and groups administered by a superuser on a centralised server. The server is not trusted to maintain the integrity of the filesystem, but it is trusted to maintain confidentiality.

Busca et al.'s Pastis [66] is a read-write distributed filesystem based on the Past DHT. Like Ivy, it relies on users signing filesystem transactions which are backed by a content-addressed store. Unlike Ivy, it allows servers to update the *inode blocks* that point to content blocks. This mechanism is more reminiscent of OceanStore, but unlike OceanStore, the updates are based on certificates rather than public ACLs. This allows content owners to distribute authority to modify content without publishing information about potential writers.

Like Venti, Ivy, CASPER and SUNDR, Pastis deals with plaintext. Servers are trusted to maintain confidentiality properties; keeping secrets from servers requires extra-filesystem encryption: "users may encrypt files' contents to ensure data confidentiality if needed" [66].

The Tahoe filesystem [232] stores files as encrypted shares using Reed-Solomon codes. File contents are stored as Merkle trees of immutable blocks, as in Footlights, but files can also be mutable. Mutable files require an ephemeral public/private key pair that can be used to sign or verify the file. Directories contain lists of read-only capabilities to children in the clear, as well as encrypted read-write capabilities. Unlike Footlights, the original Tahoe design maintained a mapping of users to ciphertext files. This was necessary for accounting purposes, but the original storage provider `allmydata.com` could observe all user activity. This design was not compatible with network-level op-

Listing 4.3: The root of a Footlights profile as a MAGNET-URI.

```
magnet:?xt=urn:sha1:XH2ZD25QPHXDARTSANUHT7VF2FWIK52S
```

timisations for either privacy (e.g. Crowds, Tor) or performance (e.g. CDNs). Subsequently, Tahoe evolved into a purely software-based project that provided no storage space [335]. Tahoe now provides scripts for users to use when setting up their own backends, but there is no globally-shared infrastructure.

HydraFS is a filesystem built on the Hydra content-addressed store [225]. It is built on immutable blocks, but the similarities to Footlights end there, as HydraFS is not designed to provide the security properties which Footlights is intended to provide. HydraFS deals in plaintext, giving the CAS full visibility into block content and references and permitting server-side garbage collection.

The MAGNET-URI project encourages file-sharing software to name content with URIs that are both URNs and relative URLs [261]. This scheme is entirely compatible with Footlights: Alice's profile in Listing 4.1 on page 74 could be encoded as the MAGNET URI in Listing 4.3, which is suitable for parsing as an HTTP query or as a URN. In this way, existing file-sharing networks could be used to provide transport for Footlights data. The availability problems of P2P networks would still apply, but the transport would cost no money.

4.10 SUMMARY OF CONTRIBUTIONS

In this chapter, I have described how untrusted commodity storage can be used to provide a secure, scalable, available distributed filesystem for social applications. I exploit commodity storage for availability and cryptography for end-to-end confidentiality and integrity.

The system organises immutable content in a content-addressed store, requiring constant-size communication to share arbitrary-sized snapshots. Mutable names refer to signed, versioned snapshots of content, allowing the construction of a writable filesystem with semantics that will be familiar to application developers.

The cost of running this storage system, based on public information about centralised OSNs and cloud storage providers, is expected to be low: on the order of one US dollar per user-year. Such a system could be supported by direct payment or a privacy-preserving advertising system.

This end-to-end system is compatible with network-level optimisations for both privacy and performance. Most users are expected to prefer performance optimisations to privacy optimisations: such users' data will be safe from confidentiality threats, but traffic analysis could reveal access patterns of a user's IP address. Users seeking assurance of anonymity can use apparently-random padding bits in overt Footlights blocks to carry covert messages over a *perfectly unobservable* communications channel.

Footlights' scalable storage system provides a storage medium decoupled from privacy and performance optimisations, allowing users to choose what degree of each they would like. This system can be used as a substrate for the social applications described in Chapter 5, "Distributed Social Applications".

5 Distributed Social Applications

It's the apps, stupid!

Peter O'Kelly, Lotus Development Corp. 1993 [354]

Platforms matter. Popular, centralised online social networks (OSNs) are not just websites, they are ecosystems. Facebook, Google+, Twitter, LinkedIn, even Yahoo! provide APIs which application developers use to enhance users' online social networking experience. If Zittrain's hypothesis [248] is to be believed — and I believe that it is — then the key driver of technological staying power is *generativity*, the ability to create new and unexpected things out of existing technology. The developers of Compuserve and Nokia OS produced compelling features, but they could never compete with all of the creativity wielded by the world's collective developers — generative platforms won.

Today's centralised OSNs have applications, but as Chapter 3 showed, those applications have been used to share private user information beyond the intent of users. Distributed social networks have been proposed as a remedy to the privacy shortcomings of traditional OSNs but almost all lack support for third-party applications (see Section 5.5). Without third-party applications, the OSN's developers must create all functionality that users might ever desire. This is analogous to requiring Facebook's in-house developers to develop all popular applications such as FarmVille [334], Spotify [331] and Draw Something [332], some of which use Facebook as little more than an authentication provider.

In this chapter, I present the Footlights social application platform. This platform demonstrates the practicality of unprivileged distributed social applications that are developed in a similar way to today's client-server social applications. Footlights allows developers to enhance the social networking experience with applications that are constrained by users' choices. I will motivate the features provided to Footlights applications by considering the needs of the lately-popular Draw Something application [332].

This application is a game in which players alternate drawing and guessing what the other player has drawn. The game is asynchronous: one player draws a picture, then sends it to a Facebook friend by uploading the picture to a third-party cloud service and notifying the friend that it is their turn. Later, the other player downloads the drawing, guesses what it is then takes a turn at drawing. The essential characteristics of this application are an ability to interactively input some state (the drawing or the guess) from a user, upload it to a storage service, share it with another user, keep statistics and display basic information such as a name and a thumbnail for each player.

Footlights provides the ability to perform these actions. Although I have not replicated the Draw Something app, I will demonstrate the ability of the Footlights API to provide the required features by presenting several basic proof-of-concept applications that run on the Footlights platform.

On the Footlights platform, unprivileged applications run locally on users' computers, where they can be confined and their behaviour can be observed (Section 5.1). Applications are written in familiar languages: UI components are written in ECMAScript 5 using a DOM-like API (Section 5.2) and back-end components can be written in any JVM-compatible language (Section 5.3). The security API presented to applications is designed to make the easiest way of implementing an application a *privacy-preserving* way; the ability to construct functional, privacy-preserving applications is demonstrated with several proofs of concept (Section 5.4). Finally, I conclude with a discussion of related attempts to provide alternative OSNs and techniques for implementing privacy-preserving social applications (Section 5.5).

5.1 MODEL OF COMPUTATION

In order to confine application behaviour, preventing applications from sharing user information more widely than users desire, it is necessary to choose a different model of computation from existing OSNs. This model is radically different because it returns to an older model: application code executing in a local context, operating on local data. This model allows Footlights — unlike today's OSNs — to take advantage of useful protection technologies that have been developed over the past decades.

The last several decades have seen many security technologies transferred from experimental, research and military/intelligence contexts into consumer software. This transition has increased the protection of users' private information from malicious applications, but today's OSNs cannot leverage these advances in data protection because they use an incompatible model of computation. Instead, they are only compatible with Digital Rights Management (DRM) schemes.

5.1.1 Securing user data

Over the past decades, several security technologies from the realms of research have been adopted by commodity operating systems to protect users' private data. In 2010, I co-authored a paper with Joseph Bonneau and Frank Stajano describing how this occurs in several application markets [12]; this section will summarise a few relevant highlights from that work.

Mandatory Access Control (MAC) is the imposition of a system-wide security policy on the behaviour of users and software. It originated in the US military [249] and intelligence community [217], where it was used to enforce Multi-Level Security (MLS) [98] and integrity [47] policies. More recently, however, MAC has appeared in widely-deployed commodity operating systems such as Linux [163], FreeBSD [226] and Mac OS X [325]. In these operating systems, MAC is used to confine applications, preventing a malicious or compromised application from accessing users' private information. This protection of user data is possible because the OS kernel which mediates all application I/O is the same kernel that is responsible for enforcing the security policy: no communication occurs outside its purview.

Similarly, capability systems have developed from a 1970s research agenda, starting with work by Needham et al. [191] and Feiertag and Neumann [102], to inspire the design of modern programming languages such as Java and EC-

MAScript 5 [117, 176, 177]. In 2010, Robert Watson, Ben Laurie, Kris Kennaway and I incorporated capability concepts into the FreeBSD operating system, improving the security properties of common applications such as `gzip` and Chromium through the Capsicum project [1, 9, 17].

Capabilities are shareable, unforgeable tokens of authority. Examples include references in object-oriented languages such as Java or JavaScript: software cannot invent new references to existing objects, it must acquire a reference from something that already holds one. When combined with a least-privileged — or *sandboxed* — model of execution, software can be run without access to any data that the user has not explicitly shared with it. For instance, in the CapDesk environment [227], a text editor would have no authority to open text files. Instead, it would request that the operating system draw a dialogue box prompting the user to choose a file; only a capability to this file would be returned to the editor. This technique generalises to the concept of *powerboxes* — dialogues that delegate user authority to applications via trusted operating system components as described by Yee [241]. Applications can only leverage the authority of the user if that user makes an explicit expression of intent.

Private user information can be protected by Mandatory Access Control, capability discipline or other techniques such as Mysers and Liskov's Distributed Information Flow Control [185] — dubbed "Sticky Policies" by Pearson and Mont — but all of these techniques have a common requirement: a trusted monitor that mediates all activity in the system. This aligns well with traditional software execution environments — desktops, mainframes and mobile phones — but it does not fit well with the structure of current OSN application platforms. In today's OSNs, applications perform computation on private information on developer-owned computers, outside the control of both users and the OSN itself. This precludes the use of any of the useful advances outlined above for the protection of user information. Instead, protecting user data from today's social social applications resembles the problem of Digital Rights Management (DRM).

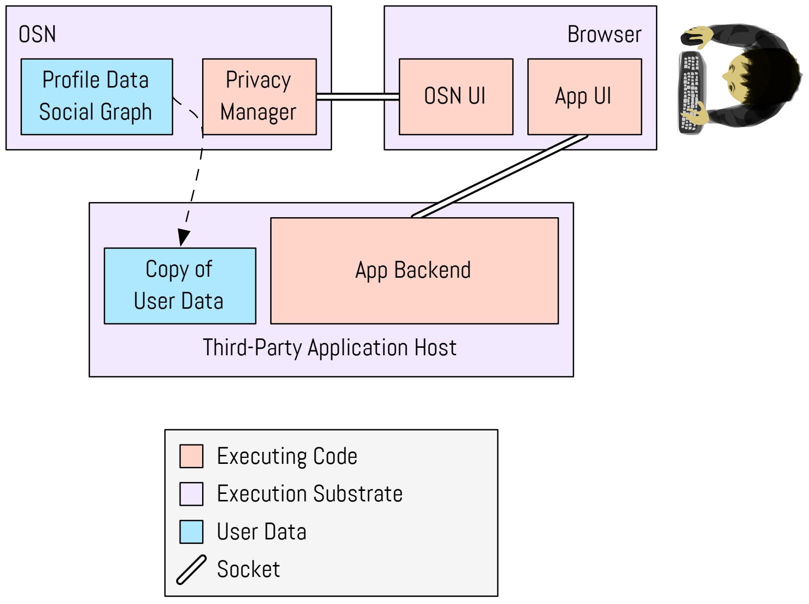


Figure 5.1: Today's OSN as an application platform.

5.1.2 Social DRM

Today's OSN applications are structured as shown in Figure 5.1: users interact with both the OSN itself and third-party applications through a Web browser. Applications request information about users directly from the OSN and may be permitted to store copies of user data [285].

OSNs may impose terms and conditions on application developers, but technical enforcement of systemic security properties is impossible, since user data is transmitted from a platform that the OSN controls (the OSN's data centre) to one that it does not (the application host). The lack of OSN-controlled infrastructure underneath the application means that techniques such as Mandatory Access Control, Information Flow Control and capability-oriented sandboxing cannot be applied to these applications.

Instead, this model of computation resembles digital rights management (DRM). In this model, sensitive information is transmitted to a platform running untrusted software; the untrusted software is permitted to use the information for one purpose (e.g. displaying a video) but not others (e.g. copying the video). DRM often takes the form of distributing copyrighted information in encrypted form separately from the relevant decryption key. Keys may be distributed to authorized players via broadcast encryption [53, 104] or even Virtual-Machine–based techniques [144]. Hardware players may incorporate self-attesting trusted components to manage decryption keys, but software-based players must, at some point, hold decryption material in memory that is visible to the computer's owner. Fundamentally, a principal cannot both possess a decryption key and also *not* possess the decryption key, so DRM schemes that do not rely on trusted hardware must instead rely on a high subversion effort. That is, if the effort required to extract a decryption key is very high, then unauthorized use may be limited for a period of time.

Most DRM schemes developed by major industries to protect copyrighted content eventually fall prey to the distributed effort of individuals. From the weak cryptography employed by DVDs [357] to the stronger cryptography employed by HD-DVD and Blu-Ray [355] to Blu-Ray–specific virtual machine techniques [341], no DRM scheme has long withstood scrutiny. Once a single key is recovered, all copies of a physical disc can be decrypted.

It might be possible for social application platforms to employ a DRM-like protection scheme: private user data could be transmitted encrypted to application hosts, which are permitted to use the data for providing service to the user but not for selling the user's data to third-party aggregators, as in §3.1.3. The question is, what OSN would implement such a scheme, and if it did, what is to be done with the decryption key?

Unlike major motion-picture studios, OSN providers have little incentive to keep user information from application developers. Users have an interest in keeping their private information private, but the proposals for users to encrypt, permute, scramble or otherwide hide information within an unmodified OSN [37, 127, 165, 167, 223] fail to protect the social graph from either the OSN provider or applications (see Section 2.3). If users did employ a DRM-like scheme, they could not make use of social applications in conventional OSNs without providing the relevant keys, but by providing such keys, all protection would be lost. If the protection scheme required trusted hardware, as in Conrado et al.'s personal DRM scheme [78], it would introduce startup and maintenance costs but still be vulnerable to the attacks employed against com-

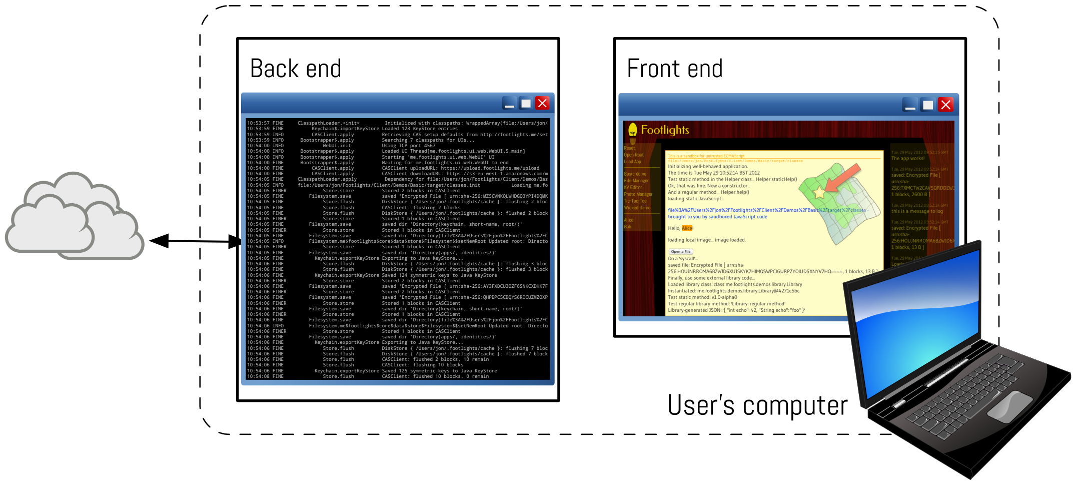


Figure 5.2: Footlights applications run locally.

mercial DRM. Finally, unlike the motion-picture scenario, users would have no means to tell whether or not the DRM scheme were even working: whereas so-called "pirates" upload cracked videos to public file-sharing services, malicious application developers sell user data privately [282].

Thus, the only known protection mechanism that is applicable to the OSN model of computation — DRM — would be ineffectual. Rather than attempting to employ social DRM, Footlights adopts a different model of computation.

5.1.3 Footlights

Footlights demonstrates that a new model for social applications is possible. This model fuses new, explicit expressions of user intent with a traditional model of computation: applications execute on users' computers, operating on user data stored on those same computers. This does not preclude highly available, scalable shared storage: Chapter 4 describes how distributed cryptography can be combined with centralised storage to provide secure, globally-available distributed filesystems. This model is about allowing applications' computation and communication to be *confined* and *observed*.

Applications running on the Footlights platform are split into two parts, as shown in Figure 5.2: a UI component suitable for interactivity and a backend component suitable for richer computation. Both are written in familiar languages such as JavaScript in the UI front-end and Java, Scala or any other language that compiles to JVM bytecode in the back-end. It is possible for applications to exist as purely one or the other, but most full-featured applications will require both. All execution is local: central servers only store encrypted blocks (see Chapter 4).

Local execution allows applications to be *confined*: they run in an unprivileged environment, starting with no authority to access user information or communicate with other applications. Application confinement is, in computer science terms, a very old concept: its "meta-theory" was described by Lampson in 1969 [152]. Nonetheless, as Chapter 3 demonstrated, applications in today's OSNs are unconfined: once user information has been shared with them, no technical restrictions are placed on the flow of that information. User information is *public* according to the definition of "Public" on page 15. In Footlights, confinement is based on existing Web and Java Virtual Machine (JVM) technologies. Applications can be granted capabilities to user data or communication sockets, but many operations can be performed indirectly, without privilege.

Local execution also allows applications to be *observed*: it is possible to check them at runtime for undesired behaviour, e.g. leaking private user data. Users can benefit from this observability even if they do not do the observation themselves. Experts can observe the behaviour of applications and share their findings with others, describing the behaviour of particular applications that have well-defined names (Section 4.4) and can thus can be reliably identified as "the same application code that I am running".

This architecture allows applications to perform arbitrary computation, but interaction with users, their data and the outside world can be strictly controlled. It also reflects a natural alignment with existing computation patterns: an application like Draw Something uses a centralised OSN to connect users to each other, but computation and user interaction is done locally. The Footlights architecture reinforces this natural separation of concerns.

A more detailed view of this front-/back-end split is shown in Figure 5.3 on the next page. Each of the two components runs atop a trusted substrate. In the front end (Section 5.2), the substrate is a collection of ECMAScript objects

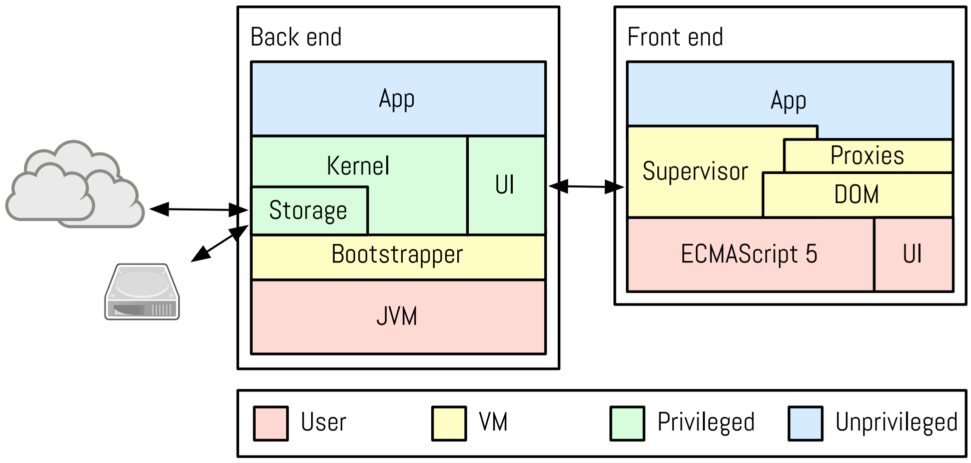


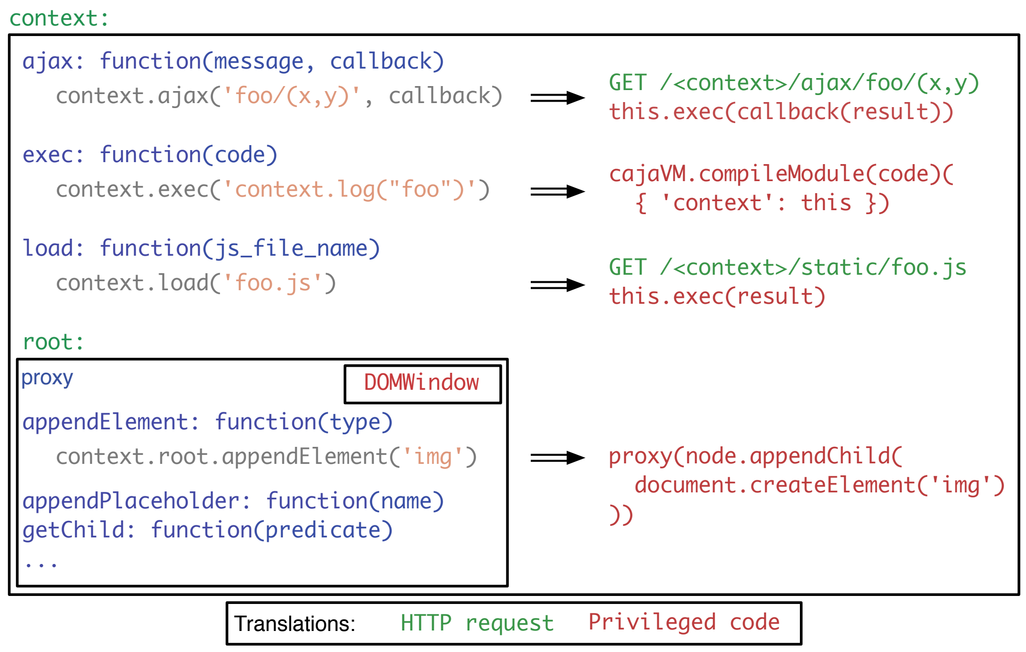
Figure 5.3: Applications are split into front-end and back-end components.

collectively called the supervisor. The supervisor proxies application access to standard Web technologies such as asynchronous JavaScript (Ajax) and the Web browser's Document Object Model (DOM). In the back end (Section 5.3), the substrate is JVM bytecode that confines application bytecode, mediating access to resources such as user information and local files. Communication between application components is mediated by the supervisor and the kernel.

5.2 FRONT END

The primary Footlights UI is a Web front-end. Other platforms could also be supported — I have explored Swing and Android UIs — but the current target is the Web browser. This environment is designed to be familiar to users of today's centralised OSNs.

The Footlights web front-end uses the features of ECMAScript 5 [251] and Google Caja [351] to confine untrusted application code. ECMAScript 5's *strict mode* provides an environment that is amenable to sandboxing and capability-oriented delegation [178]: access to global variables is restricted and shared objects can be made read-only. Whereas ECMAScript 3 was described as "one of the hardest [object-oriented] languages to secure" by Mark Miller, a Caja developer and member of the ECMA standards committee, ECMAScript 5 is "one of



```
context:
ajax: function(message, callback)
   context.ajax('foo/(x,y)', callback)      ⟹   GET /<context>/ajax/foo/(x,y)
                                                 this.exec(callback(result))

exec: function(code)
   context.exec('context.log("foo")')       ⟹   cajaVM.compileModule(code)(
                                                   { 'context': this })

load: function(js_file_name)
   context.load('foo.js')                   ⟹   GET /<context>/static/foo.js
                                                 this.exec(result)

root:
proxy                            DOMWindow
appendElement: function(type)
   context.root.appendElement('img')        ⟹   proxy(node.appendChild(
                                                   document.createElement('img')
appendPlaceholder: function(name)                ))
getChild: function(predicate)
...
```

Figure 5.4: An application's front-end context.

the easiest [object-oriented] languages to secure" [177].

Based on this foundation, Footlights executes application code within a sandboxed context (§5.2.1). Application code starts with access only to a security API that it can use to make requests of a privileged supervisor (§5.2.2). This API has been designed to closely approximate today's web UI APIs.

5.2.1 UNPRIVILEGED APPLICATION CONTEXT

Every application runs within its own unprivileged execution context. This context has no access to global browser objects such as `document` or `window`. Instead, it is provided with one read-only reference, `context`, that provides a security API partly illustrated in Figure 5.4. Through this security API, application front-ends can manipulate a delegated portion of the Footlights UI or communicate with their respective back-ends.

5.2.1.1 VISUAL CONTAINER

On instantiation, an application front-end is assigned a visual container. This container is represented by the Web browser as a DOM (Document Ob-

Listing 5.1: Creating a child node with the standard DOM API.

```
var node = document.createElement('img');
container.appendChild(i);
```

Listing 5.2: Creating a child node with the Footlights proxy API.

```
var proxy = container.appendElement('img');
```

Figure 5.5: Creating a UI element with the DOM and Footlights APIs.

ject Model) object; the DOM represents a web page as a tree of objects that can be manipulated to change the page's appearance. The Footlights web UI supervisor provides each application with a proxied DOM object that represents the root of the application's visual container. This proxy is shown as the `root` object in Figure 5.4 on the preceding page.

Proxying the application's visual container allows applications to create or manipulate child nodes without access to the global `Document` object. In the DOM specification, every document (HTML or XML) has exactly one `Document` object, to which every document element (e.g. HTML tag) has a reference, `Node.ownerDocument` [254]. Access to the `Document` object, also usually provided to JavaScript through a global variable `document`, implies an ability to access any element in the document. Applications must clearly not be given this privilege. However, access to the `Document` is also required to allocate new document elements, through factory methods such as `createElement()` and `createTextNode()`. The Footlights web supervisor allows applications to create UI elements without access to the `Document` object by creating a proxy that wraps a DOM object, including access to the `Document` object. This allows applications to create arbitrary UI components without being able to interfere with — or even name — UI components outside of their visual sandboxes. The difference between using the DOM API and the Footlights proxying API to create child elements is shown in Figure 5.5.

5.2.1.2 INTRA-APPLICATION COMMUNICATION

An application's front-end can also communicate with its back-end via the `context.ajax(message,callback)` method. This allows a front-end to send asynchronous JavaScript (Ajax) requests to its back-end.

Listing 5.3: Ajax request from an application context.

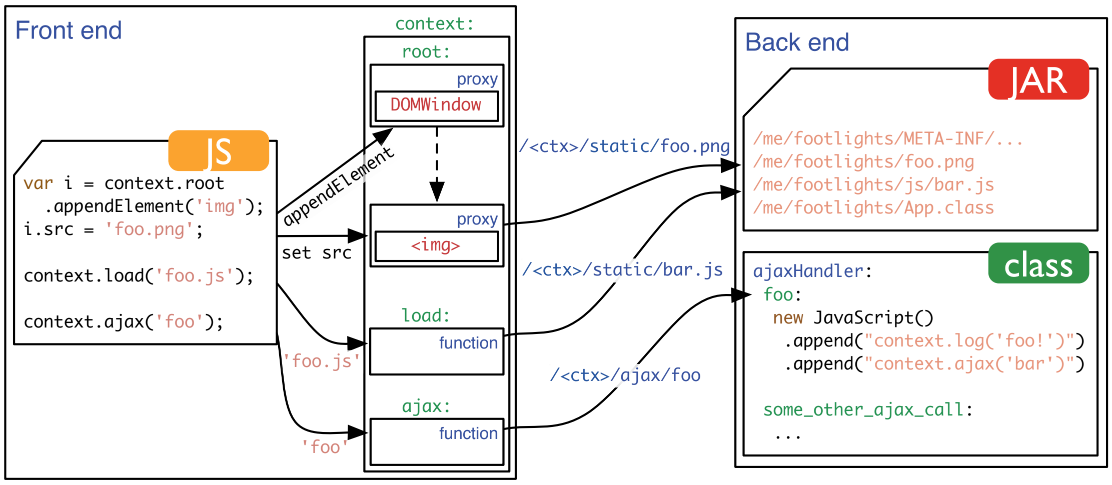
```
context.ajax('foo', function callback(response) {
    context.log('Received response: ' + response);

    // Expecting a JSON-encoded response.
    context.log('Protocol v' + response['protocol'].version);
    context.globals['foo'] = response['x'];
    context.globals['bar'].style.opacity = response['y'];
    ...
  }
);
```

Ajax requests from all active front-ends are multiplexed over a single Ajax channel between the supervisor and the Footlights back-end. If the application calls `context.ajax('foo')`, as in Listing 5.3, that request is translated by the supervisor into a request for `'/<context_name>/ajax/foo'`. This request is handled by the back-end as described in §5.3.1.4.

The asynchronous response may be in the form of code or data. In either case, the front-end making the Ajax request can specify a callback to handle the response. If the response is code, it will be passed as a string to the callback function; the front-end can execute it later within the sandbox using `context.exec()`. No callback needs to be provided to handle code; the default action is to execute the ECMAScript within the sandbox using `context.exec()`. If the response is data, a callback is required to interpret the data, as in Listing 5.3. This data will be encoded according to the JavaScript Object Notation (JSON) [258].

Application front-ends can also receive asynchronous events from their back-ends. These events are ECMAScript and are executed within the application context. Asynchronous events can be used to notify the front-end that e.g. the opponent in a game has taken a turn and the UI needs to be refreshed.



```
var i = context.root
  .appendElement('img');
i.src = 'foo.png';

context.load('foo.js');

context.ajax('foo');
```

```
/me/footlights/META-INF/...
/me/footlights/foo.png
/me/footlights/js/bar.js
/me/footlights/App.class
```

```
ajaxHandler:
 foo:
  new JavaScript()
   .append("context.log('foo!')")
   .append("context.ajax('bar')")

 some_other_ajax_call:
  ...
```

Figure 5.6: Communication over a multiplexed HTTP channel.

Through Ajax requests and asynchronous events, application front-ends can communicate with code running in the back-end. Front-ends can also communicate with the Footlights back-end in order to serve static content such as images or script files, as described below.

5.2.1.3 Static content

Applications can also load static files such as images and ECMAScript files. These files can be stored in either the application file itself or the Footlights CAS. The back-end handling of these requests is described in §5.3.3.

Applications are distributed as JAR (Java ARchive) files (see §5.3.1). These files can contain code to execute on the Footlights back-end, but they can also contain static content such as images and scripts. Images are loaded by assigning to a proxied DOM element's `src` property; the proxy translates this request into an HTTP request for the path `/<context_name>/static/<filename>`. This is an example of *API taming* as described by Stiegler, Miller, Wagner and Tribble [219, 227]: providing a safe subset of a commonly-used API rather than developing completely new APIs from scratch. Assigning to an image's `src` property corresponds exactly to the standard DOM API, but with the caveat that the Footlights supervisor will reject path names containing `".."`.

Images from the Footlights CAS can be loaded by assigning a URI to the `filename` property of a DOM proxy instead of `src`. This allows developers to package images separately from code and — if using indirect names as in Section 4.4 — change the visual appearance of applications after release without modifying any code.

Script files are executed by explicitly calling the `context.load()` method: this retrieves the script's content from the application's JAR file and executes it with `context.exec()` as in the Ajax case above. This allows application developers to write JavaScript files with conventional tools and examine their loading using conventional Web debuggers.

These three classes of functionality — UI, Ajax and static content — are provided to unprivileged applications by the privileged supervisor described next.

5.2.2 PRIVILEGED SUPERVISOR

The supervisor portion of the web UI is made up of approximately 200 lines of ECMAScript that provide Ajax and sandboxing functionality, initialise the root context for the top-level sandbox and import several Google Caja script files (approximately 1,300 lines of uncompressed ECMAScript, some of which are workarounds for incomplete browser implementations of ECMAScript 5).

The initialisation of the UI supervisor is shown in Listing 5.4 on the next page. The `root` domain is a privileged security domain that can access the global collection of sandboxes and whose root DOM proxy represents the top-level `div` containing the entire UI. However, this domain is run in a sandbox like any untrusted application: it acquires elevated privileges when it is explicitly passed powerful references in lines 27–35 of Listing 5.4 on the facing page.

One of the functions provided by the supervisor is the proxying of Ajax, static and CAS file requests. An application context's `ajax` method is actually a closure set up by the supervisor that does two things: it builds a request URL out of the context's name and the parameter supplied by the application, then forwards the resulting URL to the supervisor's low-level Ajax function. This is how e.g. a `context.ajax('foo')` call is translated into the HTTP request `/<context name>/ajax/foo`, allowing one Ajax channel to be multiplexed for use by several applications. Similar closures are used to translate the application requests for static content described in §5.2.1.3 into requests that can be handled

Listing 5.4: Supervisor initialisation.

```
'use strict';
initSES(window, whitelist, atLeastFreeVarNames);

(function init()
{
  // Set up logging (nice user-visible log box).
  var status = document.getElementById('status');
  status.log = function logToStatusDiv(t) {
    var update = document.createElement('p');
    this.insertBefore(update, this.firstChild);
    var dateStamp = document.createElement('div');
    dateStamp.appendChild(document.createTextNode(new Date().toUTCString()));
    dateStamp.className = 'timestamp';
    update.appendChild(dateStamp);
    update.appendChild(document.createTextNode(t));
  };

  function log(message) {
    console.log(message);
    status.log(message);
  }

  var rootContext = sandboxes.wrap('footlights', log);
  rootContext.root = proxy(document.getElementById('root'), rootContext);

  // The root context has no inherent privileges: pass special things in.
  rootContext.globals['sandboxes'] = sandboxes;
  rootContext.globals['window'] = window;
  ['content', 'status', 'launcher'].forEach(function(name) {
      rootContext.globals[name] = proxy(
        document.getElementById(name), rootContext
      );
    }
  );

  // A function which sets up a channel for asychronous events.
  function setupAsyncChannel() {
    setTimeout(function openAsyncChannel() {
      rootContext.ajax('async_channel');
    }, 0);
  }

  rootContext.globals['setupAsyncChannel'] = setupAsyncChannel;

  sandboxes['footlights'] = Object.freeze(rootContext);

  rootContext.ajax('init');
})();
```

by the Footlights back-end as described in §5.3.3.

Another function of the supervisor, shown in lines 36–43 of Listing 5.4 on the previous page, is to set up a channel over which asynchronous events can be forwarded to application front-ends. This is done by sending a privileged Ajax request to the `/async_channel` path, whose back-end handler blocks until there are events to be dispatched (see Section 5.3).

The supervisor also provides the *placeholder* service, which allows applications to insert private user information into the UI indirectly, without the ability to actually read the data itself. This service, inspired by Felt and Evans' "privacy by proxy" design [103], is conceptually similar to the original Facebook Markup Language (FBML). FBML once allowed application developers to insert SGML tags such as `<fb:name id="[$id]">` into their UIs. This provided a performance-enhancing indirection: Facebook could pre-populate FBML fields when serving them to clients. This indirection was deprecated in 2010 [288] and disabled in 2012 [298]. More details about FBML, as well as how it fits within the larger narrative of Facebook application privacy, can be found in §3.1.3.

The use of placeholders is shown in Figure 5.7 on the facing page, which illustrates two ways of displaying the user's name in an application UI. The first, in Listing 5.5 on the next page, is to request access to the user's name via the application back-end, which can make a request to Footlights' back-end security API, then to append that information to the UI itself. Along the way, however, the application has learned information about the user that it does not need to know, opening the door to potential misuse. The code in Listing 5.6 on the facing page, on the other hand, uses the `context.appendPlaceholder` method, causing the supervisor to issue a `fill_placeholder` Ajax request on the application's behalf. The supervisor then places the user data into a proxied element within the sandboxed UI without leaking any private information.

A second service provided by the privileged supervisor is the *powerbox* service. Traditional dialogue boxes become powerboxes when they are used to confer authority to sandboxed applications, as described by Stiegler, Miller and Yee [219, 241]. For example, the dialogue shown in Figure 5.8 on the next page is not drawn by the currently-running application: applications have no au-

Listing 5.5: Displaying the user's name without placeholders.

```
// Create UI element to hold the user's name.
context.globals['name_element'] = some_ui.appendChild('span');

// Ask the back-end to request user name from security API.
context.ajax('get_value/self.name', function callback(json) {
  // Append the retrieved value to the UI element.
  context.globals['name_element'].appendText(json.response);

  // Potentially do something malicious with the private data?
  context.globals['sell_to_rapleaf']('name', json.response);
});
```

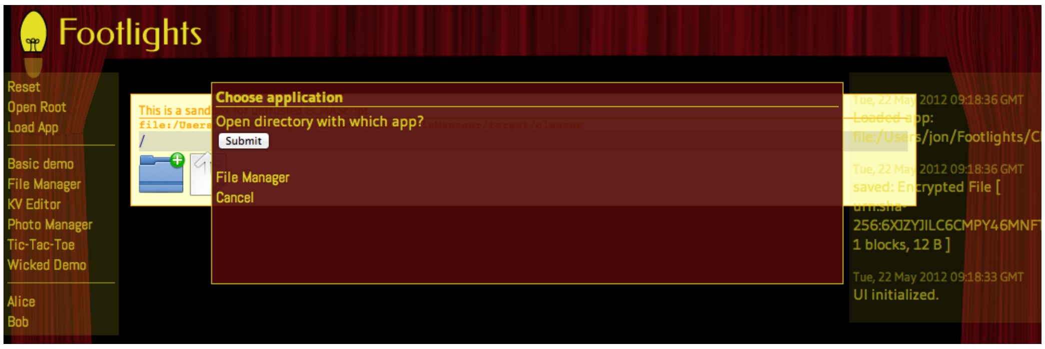
Listing 5.6: Displaying the user's name with placeholders.

```
some_ui.appendPlaceholder('self', 'name');
```

Figure 5.7: Two ways of displaying the user's name.

thority to enumerate other applications. Rather, this dialogue is drawn with the authority of trusted software (the Footlights core) to ask the user what authority (sending a directory to an application) should be granted to untrusted software. The provision of the powerbox service is described in §5.3.4.2.

Like the placeholder service and multiplexed Ajax service, the powerbox



Choose application
Open directory with which app?

This is a sand
file:/Users
/

Tue, 22 May 2012 09:18:36 GMT
Loaded app:
file:/Users/jon/Footlights/C

Tue, 22 May 2012 09:18:36 GMT
saved: Encrypted File [
urn:sha-
256:6XJZYJILC6CMPY46MNF
1 blocks, 12 B]

Tue, 22 May 2012 09:18:33 GMT
UI initialized.

Figure 5.8: A Footlights powerbox.

service depends on communication with the Footlights back-end. That back-end, which also hosts application back-ends, is described next.

5.3 Back end

The back-end of a Footlights application is hosted on a Java Virtual Machine (JVM) based platform. The JVM supports a rich, fully-featured programming environment in which applications can perform efficient byte-level computation. Applications can be written in any language that targets the JVM: core Footlights interfaces are specified in Java, the lowest-common-denominator language for the JVM, although some support libraries are also provided in the Scala language [193]. The JVM also provides a mechanism for executing untrusted, compiled bytecode in a constrained environment; Footlights uses this mechanism to confine application code.

5.3.1 Applications

Application back-ends run as untrusted bytecode on the JVM. They are confined as described in §5.3.2 to prevent them from directly interacting with user data or the outside world. These interactions are all mediated by the security API [11] described in Section 5.3.4.

In this section I describe how applications are distributed (§5.3.1.1) and initialised (§5.3.1.2) and provide services to other applications (§5.3.1.3).

5.3.1.1 Distribution

Footlights applications are distributed as JAR (Java ARchive) files in the Content-Addressed Store (CAS) described in Chapter 4. An application JAR must contain a manifest file in the conventional location `META-INF/MANIFEST.MF` that describes the basic structure of the application. An example of a Footlights application manifest is shown in Listing 5.7 on the facing page. This manifest declares that the application's main class is `me.footlights.demos.good.GoodApp`, which can be found in the conventional location within the JAR file (`/me/footlights/demos/good/GoodApp.class`).

Listing 5.7: An example manifest file for a Footlights application.

```
Manifest-Version: 1.0
Class-Path: http://footlights.me/demo/library-v1
Footlights-App: me.footlights.demos.good.GoodApp
```

Listing 5.8: Initialisation of a sample Footlights application.

```
package me.footlights.demos.good;

import scala.Option;
import me.footlights.api.*;

public class GoodApp extends me.footlights.api.Application
{
  public static GoodApp init(KernelInterface kernel,
      ModifiablePreferences prefs, Logger log)
  {
    log.info("Loading " + GoodApp.class.getCanonicalName());
    return new GoodApp(new DemoAjaxHandler(kernel, log));
  }
  /* ... */
  private GoodApp(AjaxHandler ajax)
  {
    super("Basic demo");
    this.ajax = ajax;
  }
}
```

The manifest also declares that the application depends on a library named by the URL `http://footlights.me/demo/library-v1`. Like standard JAR files, this dependency is declared using the `Class-Path` entry name. Unlike standard JAR files, however, the dependency is not specified with a local filename. Instead, it is named with a Footlights CAS name. The dependency can be on a JAR file which hashes to an exact value; such a name is given as the hash of an immutable file, which implicitly provides integrity protection (Section 4.3). Alternatively, the dependency can be named by a mutable URL that refers to such a name indirectly (see Section 4.4). This approach provides more flexibility — the app will always use an up-to-date library — but the integrity of the referenced URL must be protected through other means.

5.3.1.2 Initialisation

This main class must have a static `init` method like that in Listing 5.8 on the previous page. `init` is to a Footlights application what `main` is to a C application: a linkage point where execution can begin. The parameters of `init` are three references: the Footlights kernel (Section 5.3.4), a persistent key-value store (§5.3.5) and a `java.util.logging.Logger` instance for logging messages.

On invocation, `init` must return a subclass of the Footlights-provided abstract class `me.footlights.api.Application`, shown in Listing 5.9 on the facing page. This class provides three optional services that may be overridden: a file-opening service and a directory-opening service, both described in §5.3.1.3, as well as an Ajax handler, described in §5.3.1.4.

5.3.1.3 File services

Applications can provide file- or directory-based services to other applications by overriding the methods `Application.open(File)` and `Application.open(Directory)` in Listing 5.9 on the next page. This allows other applications to share content with them, directed by the user according to the techniques described in §5.3.4.2. An example of how this sharing can be used in practice is illustrated by the File manager application in §5.4.5.

Future versions of Footlights might also allow applications to advertise the services they provide in a registry, using a description like a MIME `Content-`

Listing 5.9: The abstract Application class.

```
package me.footlights.api;

import scala.Option;Applications
import me.footlights.api.ajax.AjaxHandler;

public abstract class Application
{
  public Application(String shortName) { ... }

  /** Open a file from another application. */
  public void open(File file) {}

  /** Open a directory from another application. */
  public void open(Directory directory) {}

  /** A handler for Ajax requests from the Web UI. */
  public Option<AjaxHandler> ajaxHandler() { ... }

  ...
}
```

`Type` [265, 257]. Such a registry would be conceptually similar to those provided by Android [329], D-Bus [164], launchd [322] and even inetd [348].

5.3.1.4 AJAX SERVICE

In order to communicate with its front-end, an application can provide an `AjaxHandler` by overriding the `Application.ajaxHandler()` method.

The abstract class `AjaxHandler` is illustrated in Listing 5.10 on the following page. It or its subclasses can respond to a request issued by the front-end with an `AjaxResponse`. This response can be a `JavaScript` object representing code to execute in the front-end's sandboxed context or a `JSON` object representing data to pass to a front-end callback. This allows the application front-end to communicate with the back-end in response to UI events, JavaScript timers, etc.

When an application is started, the back-end is first initialised, then the front-end visual context is created (§5.2.1.1). Once this is complete, the front-end supervisor initialises the UI by sending an Ajax request string `"init"` to the back-end. The response to this request is expected to provide a `JavaScript`

Listing 5.10: A simplified version of the AjaxHandler class.

```java
/** An object that can process Ajax requests. */
public abstract class AjaxHandler
{
  public AjaxResponse service(WebRequest request)
  {
    return new JavaScript();
  }

  public void fireAsynchronousEvent(JavaScript code) { /* */ }

  /* ... */
}
```

object representing the code required to initialise the UI.

When an `AjaxHandler` wants to send unsolicited messages to its front-end, it can call its own `fireAsynchronousEvent()` method. This method enqueues a `JavaScript` object to be delivered via the Web UI's asynchronous channel. As described in §5.2.2, this is a multiplexed channel, shared by all sandboxed UI contexts. This channel allows application back-ends to send event notifications to their front-ends when directories are shared with the application, long-running tasks are completed, etc.

5.3.2 Confinement

Confinement of application back-ends is based on the Java security model, which allows the JVM to run different bytecode with different permissions as described by Gong et al. [119] These permissions are assigned by the ClassLoader that loads the bytecode, so Footlights has a custom ClassLoader.

The Footlights back-end is divided into three key components: the Bootstrapper, which contains a custom ClassLoader, the Kernel, which supports applications and a UI, which manages the Footlights front-end. The Bootstrapper is responsible for loading both the Kernel and any UIs (Web, Swing, Android or other), and it assigns them the `java.security.AllPermission` when it does. This permission allows trusted code to access files, network sockets, draw to the UI, etc. subject to certain constraints discussed in the next paragraph. Any code loaded after the Kernel and UI is treated as entirely unprivileged; it cannot perform these operations. One of the few "system" privileges available to unprivileged code is the ability to read the system time; if the JVM provided a means to control this via the same privilege system, Footlights could close covert channels among malicious applications, as described by Lipner [159].

In the Java security model, privilege is not always applied automatically. Before performing a privileged operation such as opening a local file, the JVM checks the call stack to ensure that only privileged (and thus trusted) code is involved in the operation. This prevents trusted code from being abused with a *confused deputy* attack. A confused deputy, as illustrated by Hardy, is privileged code that is tricked into using its privilege in an unintended way [130]. The JVM's stack-checking scheme prevents confused deputy attacks like the *return-to-libc* attack by Peslyak (a.k.a. Solar Designer) [353]. Privileged code can, however, explicitly perform privileged operations by calling `java.security.AccessController.doPrivileged()`. This allows privileged code to assert that it is being called through well-defined interfaces and that it accepts responsibility for sanitising input appropriately. In this case, the JVM will only look at the call stack between the privileged operation and the most recent `doPrivileged` invocation by privileged code.

Footlights exposes an explicit security API to applications (Section 5.3.4), so all use of JVM privilege can be represented by a single Scala trait. This trait acts as a privilege-adding wrapper around the Footlights security API. The trait is illustrated in Listing 5.11 on the next page; it has been simplified beyond what the Scala compiler accepts in the interest of readability.

5.3.3 STATIC CONTENT

As described in §5.2.1.3, application front-ends can refer to static content located either in the application JAR file or the Footlights Content-Addressed Store (CAS). For instance, static images can be rendered as part of the sandboxed Web UI.

Application JAR files can contain static content such as images for display in the UI. When the Footlights Web UI receives a request for such content, it comes

Listing 5.11: A simplified version of Footlights' KernelPrivilege trait.

```
trait KernelPrivilege extends Footlights {
  def open(name:URI)    = sudo { () => super.open(name) }
  def openLocalFile()   = sudo { () => super.openLocalFile() }
  def open(link:Link)   = sudo { () => super.open(link) }
  /* ... */
}

object Privilege {
  /**
   * Execute a function with JVM privilege.
   *
   * The name "sudo" is meant to be evocative of privilege in
   * general; it does not refer specifically to system
   * privilege as conferred by sudo(8).
   */
  private[core] def sudo[T](code:() => T) =
    try AccessController.doPrivileged[T] {
      new PrivilegedExceptionAction[T]() {
        override def run:T = code()
      }
    }
    catch {
      case e:PrivilegedActionException => throw e getCause
    }
}
```

as a request for a path such as `/<contextname>/static/me/footlights/foo.png`. This request is parsed by the Web UI back-end, which retrieves from the kernel the application named by the given context name.

From the application, the Web UI can retrieve the application-specific Footlights `ClassLoader` used to instantiate it (§5.3.2). This `ClassLoader` can open resources from its class path, given a relative name that identifies a file within the application's JAR bundle. The static content can then be served by the Web UI on behalf of the application. In this way, applications can refer to arbitrary static content bundled with the application. Practical uses of this capability include the script files and image files essential to a web-based application UI.

Applications can also refer to static content contained in the Footlights content addressed store (CAS). Immutable content is named by its URN (Section 4.3) or indirectly through a mutable URL (Section 4.4). When the Web UI receives a request for CAS content, it simply retrieves the binary `File` from the kernel as shown in §5.3.4.1 and serves it to the application. This allows content such as image files to be stored in the CAS and displayed in the UI, reducing the amount of content that must be distributed with the application. For instance, an application can bundle static bytecode in a JAR file but rely on a mutable URL to direct users to up-to-date graphics.

5.3.4 KERNEL API

The Footlights kernel can be accessed by applications via the API shown in Listing 5.12 on the following page. This API currently demonstrates two kinds of services: operations on files (§5.3.4.1) and interactions with the user (§5.3.4.2). A more complete future API could also provide tools for synchronous collaboration among applications (§5.3.4.3).

5.3.4.1 FILES AND DIRECTORIES

As described in Chapter 4, the Footlights storage system provides a mutable shared filesystem based on immutable structures. This construct is provided to applications for two purposes. The first purpose is *compatibility*: today's applications store hierarchical information in platform-provided filesystem namespaces, so a similar Footlights offering will make the application development

Listing 5.12: Footlights kernel interface.

```
/** An application's interface to the Footlights kernel. */
public interface KernelInterface
{
  /** Save data to a logical file. */
  public Either<Exception,File> save(ByteBuffer data);

  /** Open a file by its URN. */
  public Either<Exception,File> open(URI name);

  /**
   * Open a file using a hierarchical directory namespace.
   *
   * The name given can be rooted in either a URN
   * (e.g. "urn:foo/some/path/to/file") or an
   * app-specific root (e.g. "/my/path/to/a/file").
   */
  public Either<Exception,File> open(String name);

  /** Open a mutable directory (wraps immutable dir). */
  public Either<Exception,Directory> openDirectory(String n);

  /** Open a file on the local machine (user chooses). */
  public Either<Exception,File> openLocalFile();

  /** Save data into a local file. */
  public Either<Exception,File> saveLocalFile(File file);

  /** Ask the user a question. */
  public Either<Exception,String>
    promptUser(String prompt, Option<String> defaultValue);

  /** Convenience method with no default value. */
  public Either<Exception,String> promptUser(String prompt);

  /** Share a {@link Directory} with another user. */
  public Either<Exception,URI> share(Directory dir);

  /** Open a file with another app. */
  public Either<Exception,File> openWithApplication(File f);

  /** Open a directory with another app. */
  public Either<Exception,Directory>
    openWithApplication(Directory dir);
}
```

environment familiar to developers. The second purpose of the filesystem abstraction is to provide a natural unit of *sharing*: anything that can be expressed by an application as a directory hierarchy can be trivially shared with other users or applications.

A Footlights file, shown in Listing 5.13 on the next page, is an immutable quantity of binary data. Its contents can be read via a Java `InputStream` as in the Java file API. This could be augmented in the future with Java NIO primitives [250] to allow asynchronous transfers and the performance they bring, but the current API demonstrates compatibility with existing API expectations.

Because files are based on a Content-Addressed Store (CAS), every immutable file can have a globally-unique name derived from a hash of its contents. This name, as returned by `File.name()`, is a Uniform Resource Name (URN): it identifies content rather than storage location. If the content of the file were to change, it would not be the same file any more by the CAS definition. To modify a file, an application must copy it into a MutableFile, which can be modified and then frozen into a new immutable File, or else copy the file's current contents into a mutable buffer via `File.copyContents()`. An application can then convert the modified buffer into a new immutable file via `KernelInterface.save(ByteBuffer)`.

When an application saves data as a file, the decryption key for that file is saved in an application-specific keychain maintained by the Footlights kernel. This keychain is simply a serialisable in-memory mapping to keys from the names of the blocks they decrypt. The application can later access the file by name, implicitly using this keychain to decrypt the file. Applications never access cryptographic keys: they are always managed implicitly via a directory hierarchy or the application keychain.

The interface of a directory is shown in Listing 5.14 on page 127. Every application receives its own virtual filesystem, which can be accessed by calling `openDirectory("/")` on its `KernelInterface` reference. As in traditional filesystems, a Footlights directory contains a set of entries which map an application-chosen name to a file or directory [173]. Unlike traditional filesystems, however,

Listing 5.13: A Footlights file.

```
/**
 * A logical file.
 *
 * Files are immutable; to modify a file, you must call
 * {@link mutable()}, which returns a {@link MutableFile},
 * modify that, and {@link MutableFile.freeze()} it.
 */
public interface File {
  public interface MutableFile
  {
    public MutableFile
      setContent(Iterable<ByteBuffer> content);

    public File freeze() throws GeneralSecurityException;
  }

  /**
   * The file's name.
   *
   * Client code should treat filenames as opaque identifiers;
   * they are certainly not guaranteed to be human-readable.
   */
  public URI name();

  /**
   * The content of the file, transformed into an
   * {@link InputStream}.
   */
  public InputStream getInputStream();

  /**
   * The content of the file.
   * Calling this may be unwise for large files!
   */
  public ByteBuffer copyContents() throws java.io.IOException;
}
```

Listing 5.14: A Footlights directory.

```
/**
 * A mapping from application-specified names to {@link File}
 * and {@link Directory} objects.
 *
 * A {@link Directory} is mutable from an application
 * perspective, but maps onto immutable structures behind
 * the {@link KernelInterface}.
 */
public interface Directory
{
  public interface Entry
  {
    public boolean isDir();
    public String name();
    public Either<Exception,Directory> directory();
    public Either<Exception,File> file();
  }

  /** Name of current snapshot (an immutable directory). */
  public URI snapshotName();

  // Operations on directory entries
  public Iterable<Entry> entries();
  public Option<Entry> get(String name);
  public Either<Exception,Directory> remove(String name);

  /** Files (not directories) in this directory. */
  public Iterable<Tuple2<String,File>> files();

  /** Direct sub-directories of this directory. */
  public Iterable<Tuple2<String,Directory>> subdirs();

  /** Open by relative name, underneath this directory. */
  public Either<Exception,File> open(String name);
  public Either<Ex...,Directory> openDirectory(String name);

  /** Save to this directory (direct child). */
  public Either<Ex...,Entry> save(String name, File file);
  public Either<Ex...,Entry> save(String name, ByteBuffer b);
  public Either<Ex...,Entry> save(String name, Directory dir);

  /** Create a new subdirectory (fails if already exists). */
  public Either<Exception,Directory> mkdir(String name);

  /** open() if subdir exists, mkdir() if it doesn't. */
  public Either<Exception,Directory> subdir(String name);
}
```

the underlying objects being mapped to are content-addressed, so a directory mapping is also implicitly a hash tree. At any point in time, a Footlights directory is also an immutable snapshot: updating a directory's entry set creates a new directory and notifies the directory's owner of the change.

In the case of subdirectories, the owner is the parent directory. Modifying a subdirectory results in the parent directory being notified so that it can update its own entry map to point at the new subdirectory. Notifications propagate up the tree until they reach the root directory of the working tree, which must be able to handle the update specially.

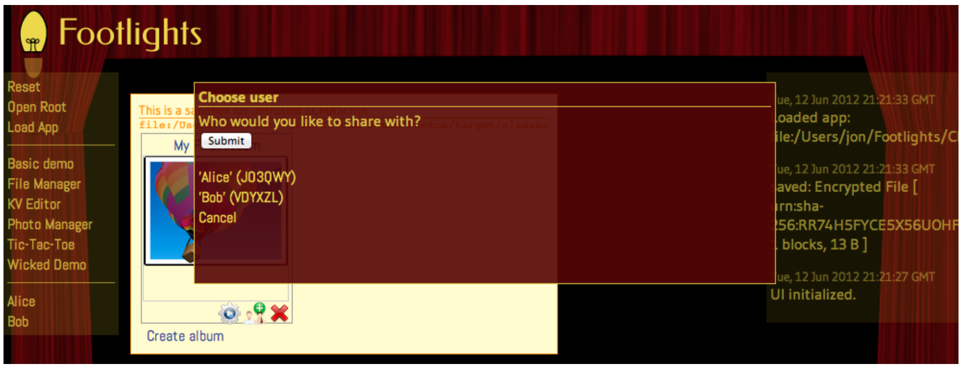
In the current implementation of Footlights, an application's virtual filesystem is a directory within a larger per-user filesystem. Change notifications from application directories propagate upwards to a single root directory kept by the user's Footlights instance. This root directory is instantiated with a callback that saves the directory's encryption key to a local keychain and then atomically saves the directory's name to a file in the local filesystem.

5.3.4.2 Powerboxes

Applications can interact with the user by manipulating the Web UI's Document Object Model (DOM) as described in Section 5.2. This allows applications to render objects and attach event handlers for e.g. mouse clicks, but it does not allow applications to ask users questions such as, "which file would you like to open?", or "which user would you like to share this photo with?"

These interactions are done via the back-end `KernelInterface` API. For instance, a photo-sharing application will need to open photos from the local filesystem. An unsandboxed application might access the filesystem, list the photos present and prompt the user with a dialogue box to choose one of them. A Footlights application has no authority to do this, however. Instead, a Footlights application's back-end can call kernel methods such as `openLocalFile()` and `saveLocalFile()`. The kernel displays a powerbox, the user chooses a file and a read-only `File` is returned to the Footlights application. In this way, the user directs a privileged component (the Footlights kernel) to delegate limited authority to an unprivileged component (the Footlights application).

Applications can also call the `KernelInterface` API in order to share con-



Choose user

Who would you like to share with?

'Alice' (J03QWY)
'Bob' (VDYXZL)
Cancel

```
ue, 12 Jun 2012 21:21:33 GMT
oaded app:
ile:/Users/jon/Footlights/C

ue, 12 Jun 2012 21:21:33 GMT
aved: Encrypted File [
rn:sha-
56:RR74H5FYCE5X56UOHF
 blocks, 13 B ]

ue, 12 Jun 2012 21:21:27 GMT
UI initialized.
```

Figure 5.9: Sharing a directory with another user.

tent with other applications or to share content with other users, resulting in a powerbox like that in Figure 5.9. In both cases, the kernel uses its privilege (to enumerate running applications or friends of the user) to present the user with options, then lets the user direct who or what the application-specified content is shared with.

5.3.4.3 SYNCHRONOUS COMMUNICATION

A third set of kernel methods would provide applications with primitives for synchronous communication. These methods have not been implemented in the current Footlights prototype.

In order to support applications with a real-time component such as gaming or Borisov et al.'s Off-the-Record instant messaging [56], a complete social application platform should provide synchronous messaging primitives. When applications running on the same host collaborate, the Footlights platform could transport JSON objects between them as it does between application front- and back-ends. When applications running on different hosts collaborate — as in the case of two users playing a real-time game — the users' Footlights clients could set up a communication channel between them using established "hole punching" techniques identified by Ford et al. [106] or Signposts as developed by Chaudhry et al. [344, 345]. Once this is accomplished, application channels can be multiplexed over the top using the same mechanisms as local application communication.

Listing 5.15: The ModifiablePreferences class.

```java
public interface ModifiablePreferences extends Preferences
{
  public Preferences set(String key, String value);
  public Preferences set(String key, boolean value);
  public Preferences set(String key, int value);
  public Preferences set(String key, float value);

  public Preferences delete(String key);
}
```

Listing 5.16: The Preferences class.

```java
public interface Preferences
{
  public enum PreferenceType {
    STRING, BOOLEAN, INTEGER, FLOAT };
  public Iterable<Map.Entry<String,PreferenceType>> keys();

  public Option<String>  getString(String key);
  public Option<Boolean> getBoolean(String key)
  public Option<Integer> getInt(String key);
  public Option<Float>   getFloat(String key);
}
```

5.3.5 Persistent key-value store

Having described the first argument to an application's `init` method — a reference to a `KernelInterface` object — I will now describe the second: a reference to a persistent key-value store.

The key-value store allows applications to store non-hierarchical information. For instance, a game can store user preferences and statistics (e.g. which game board to draw, how many games the user has won) in the key-value store, letting it do the work of type checking and serialisation.

The interface of this store is shown in Listing 5.15 (which references Listing 5.16). It currently handles only primitive types, although it could easily be extended to cover arbitrary binary content stored in the Footlights CAS. This would allow applications to eschew the file system altogether in favour of a

Listing 5.17: Manifest file for the Basic Demo application.

```
Manifest-Version: 1.0
Class-Path: http://footlights.me/demo/library-v1
Footlights-App: me.footlights.demos.good.GoodApp
```

non-hierarchical store.

Updates by applications to the Footlights key-value store are atomic, but there is no transaction mechanism yet to ensure consistency when updating multiple entries.

By providing these APIs to confined bytecode, untrusted applications can perform useful work and operate indirectly on user data. I have evaluated this claim by building several demonstration applications that exercise the Footlights APIs. These applications are described in Section 5.4.

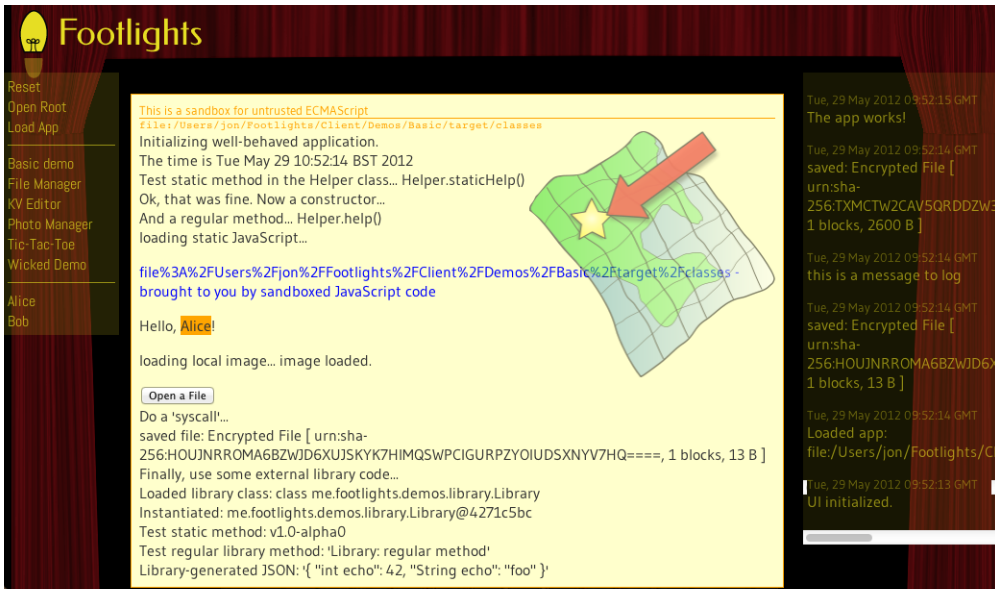
5.4 EXAMPLE APPLICATIONS

I have built several basic applications to demonstrate the functionality of the Footlights APIs (both front-end and back-end). Each is small and none are complex or production-worthy, but together they demonstrate the suitability of the Footlights APIs to support larger, more complex applications.

5.4.1 BASIC DEMO

The Basic demo application is designed to exercise several important aspects of the Footlights front- and back-end APIs. It does not constitute a practical application to satisfy a user need, but it does demonstrate how such applications could be built on the Footlights APIs. A screenshot of this application running on the Footlights platform is shown in Figure 5.10 on the following page.

On receiving its first Ajax message, `"init"` (see §5.3.1.4), the application responds with JavaScript code to do two things: add some simple text to the sandboxed UI and send another Ajax request back to the back-end. Using this work-and-callback pattern, several discrete blocks of functionality are chained



```
This is a sandbox for untrusted ECMAScript
file:/Users/jon/Footlights/Client/Demos/Basic/target/classes
Initializing well-behaved application.
The time is Tue May 29 10:52:14 BST 2012
Test static method in the Helper class... Helper.staticHelp()
Ok, that was fine. Now a constructor...
And a regular method... Helper.help()
loading static JavaScript...

file%3A%2FUsers%2Fjon%2FFootlights%2FClient%2FDemos%2FBasic%2Ftarget%2Fclasses -
brought to you by sandboxed JavaScript code

Hello, Alice!

loading local image... image loaded.

Do a 'syscall'...
saved file: Encrypted File [ urn:sha-
256:HOUJNRROMA6BZWJD6XUJSKYK7HIMQSWPCIGURPZYOIUDSXNYV7HQ====, 1 blocks, 13 B ]
Finally, use some external library code...
Loaded library class: class me.footlights.demos.library.Library
Instantiated: me.footlights.demos.library.Library@4271c5bc
Test static method: v1.0-alpha0
Test regular library method: 'Library: regular method'
Library-generated JSON: '{ "int echo": 42, "String echo": "foo" }'
```

```
Tue, 29 May 2012 09:52:15 GMT
The app works!

Tue, 29 May 2012 09:52:14 GMT
saved: Encrypted File [
urn:sha-
256:TXMCTW2CAV5QRDDZW3
1 blocks, 2600 B ]

Tue, 29 May 2012 09:52:14 GMT
this is a message to log

Tue, 29 May 2012 09:52:14 GMT
saved: Encrypted File [
urn:sha-
256:HOUJNRROMA6BZWJD6X
1 blocks, 13 B ]

Tue, 29 May 2012 09:52:14 GMT
Loaded app:
file:/Users/jon/Footlights/Cl

Tue, 29 May 2012 09:52:13 GMT
UI initialized.
```

Figure 5.10: A basic demo application.

together to form a complete demo.

In the next test block, the demo exercises the Footlights class loader. The `Helper` class, part of the Basic Demo bundle, contains both static and instance methods. These are both tested, along with `Helper` instantiation.

The next block of tests loads a static JavaScript file using the front-end method `context.load('text.js')`. This file is bundled as part of the demo's JAR file and handled by the static file loading mechanism described in §5.3.3. It is compiled by the front-end supervisor and exercises the context API. It adds UI elements, including a placeholder for the user's name and a static image (bundled with the application). This image is given event handlers that modify the visual appearance of various sandboxed UI elements when the mouse cursor moves over or off of the image or the user clicks it. The UI is also given a box which the user can click to send an Ajax request that triggers an Open File dialogue as described in §5.3.4.2.

The next test block saves a string of bytes (the ASCII representation of "Hello, world!") to a Footlights file and outputs the file's CAS name.

In the final block of tests, the demo exercises class path dependency resolution by calling code in a library. This library is declared as a dependency by the

Basic Demo manifest file, as shown in Listing 5.17 on page 131. The demo calls static and instance methods of a class called `Library`, including methods which themselves exercise Footlights API classes.

Together, these tests demonstrate that applications can perform many useful activities without any special privilege.

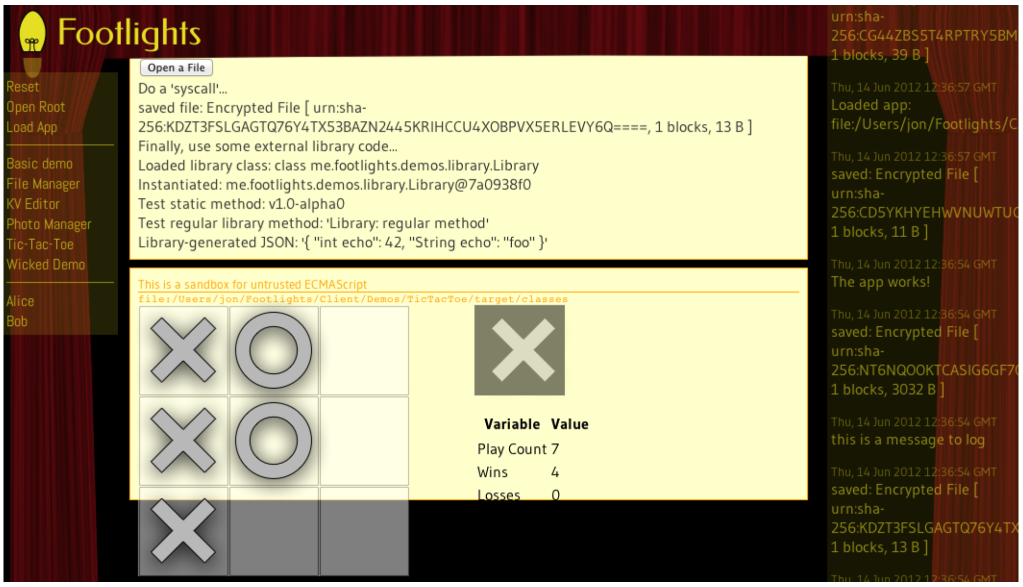
5.4.2 MALICIOUS DEMO

I have also developed a malicious application to demonstrate Footlights' confinement properties. This application tests the Footlights `ClassLoader` by attempting to load code from sources that it should not be able to access and to load its own packaged code into sealed packages. Sealed packages have been declared complete by their authors; if malicious code were loaded into them, that code would be able to access package-internal data.

The application also tests the Java security policy installed by Footlights: it attempts to access remote servers and Footlights-internal classes as well as create a new `ClassLoader`. `ClassLoader` objects are part of the Trusted Computing Base (TCB) — the computing foundation that is able to violate arbitrary security policies — because they are responsible for conferring privileges on bytecode. Untrusted applications must not be permitted to do this, as they could then confer more privileges than they possess, a form of privilege escalation.

Finally, the application legitimately loads a malicious JavaScript file to test the security API of the Web UI supervisor. This file attempts to contact remote servers, access global JavaScript objects such as `document` and `window`, write to read-only context variables and insert privileged HTML objects such as `<iframe />` and `<script/>` elements. These elements would be able to access global `document` and `window` objects, so they may not be instantiated. Instead, application front-ends can use the `context.load` method, which applies all of the protections described in Section 5.2.

The malicious demo application fails in all of these actions, demonstrating that even a rich application API can provide the confinement properties required to protect user privacy.



```
Do a 'syscall'...
saved file: Encrypted File [ urn:sha-
256:KDZT3FSLGAGTQ76Y4TX53BAZN2445KRIHCCU4XOBPVX5ERLEVY6Q====, 1 blocks, 13 B ]
Finally, use some external library code...
Loaded library class: class me.footlights.demos.library.Library
Instantiated: me.footlights.demos.library.Library@7a0938f0
Test static method: v1.0-alpha0
Test regular library method: 'Library: regular method'
Library-generated JSON: '{ "int echo": 42, "String echo": "foo" }'
```

```
This is a sandbox for untrusted ECMAScript
file:/Users/jon/Footlights/Client/Demos/TicTacToe/target/classes
```

Variable	Value
Play Count	7
Wins	4
Losses	0

```
urn:sha-
256:CG44ZBS5T4RPTRY5BM
1 blocks, 39 B ]

Thu, 14 Jun 2012 12:36:57 GMT
Loaded app:
file:/Users/jon/Footlights/C

Thu, 14 Jun 2012 12:36:57 GMT
saved: Encrypted File [
urn:sha-
256:CD5YKHYEHWVNUWTUC
1 blocks, 11 B ]

Thu, 14 Jun 2012 12:36:54 GMT
The app works!

Thu, 14 Jun 2012 12:36:54 GMT
saved: Encrypted File [
urn:sha-
256:NT6NQOOKTCASIG6GF7
1 blocks, 3032 B ]

Thu, 14 Jun 2012 12:36:54 GMT
this is a message to log

Thu, 14 Jun 2012 12:36:54 GMT
saved: Encrypted File [
urn:sha-
256:KDZT3FSLGAGTQ76Y4TX
1 blocks, 13 B ]

Thu, 14 Jun 2012 12:36:54 GMT
```

Figure 5.11: A Tic-Tac-Toe demo application.

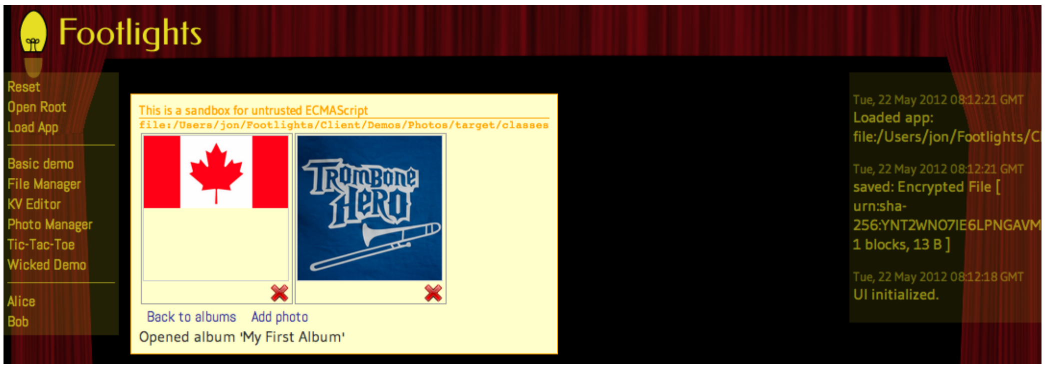
5.4.3 TIC-TAC-TOE

The Tic-Tac-Toe application depicted in Figure 5.11 demonstrates how an application developer can build an interactive game on the Footlights platform. The platform features used by the application are a strict subset of those used by the basic demo in §5.4.1, but they are used to construct a more plausible demo.

The Tic-Tac-Toe application uses the sandboxed UI features of the Web UI to draw an interactive game board. Users alternate clicking squares on this board, which sends event notifications to the application back-end via Ajax requests (see §§5.2.1.2 and 5.3.1.4). The back-end maintains the state of the game, sends game updates to the UI and records play statistics in the persistent key-value store described in §5.3.5.

The demo does not support playing games against other users: that would require the synchronous IPC features sketched in §5.3.4.3. If these features were implemented, extending the game to play against a remote rather than local opponent would be straightforward.

Tic-Tac-Toe is a straightforward game with a straightforward implementation. Building the game on the Footlights platform required 158 lines of Scala,



```
This is a sandbox for untrusted ECMAScript
file:/Users/jon/Footlights/Client/Demos/Photos/target/classes
Opened album 'My First Album'
```

```
Tue, 22 May 2012 08:12:21 GMT
Loaded app:
file:/Users/jon/Footlights/Cl

Tue, 22 May 2012 08:12:21 GMT
saved: Encrypted File [
urn:sha-
256:YNT2WNO7IE6LPNGAVM
1 blocks, 13 B ]

Tue, 22 May 2012 08:12:18 GMT
UI initialized.
```

Figure 5.12: A basic photo manager.

147 lines of Java and 93 lines of JavaScript.

5.4.4 PHOTOS

The Photos application, shown in Figure 5.12, manages a set of photo albums that can be shared with other users or applications.

The application uses loads photos from the local machine's filesystem according to user direction, as described in §5.3.4.2. The contents of these photos are converted into CAS files, which are stored in the hierarchical namespace afforded by the Footlights filesystem. Images are displayed in the Web UI using the mechanisms described in §§5.2.1.3 and 5.3.3: the application front-end simply assigns to the `filename` attribute of a proxied image element as shown in Listing 5.18 on the following page; the `name` value is derived from an `album` maintained in the back-end, as shown in Listing 5.19 on the next page.

The Photos application provides users with the ability to share albums with other users or applications. A screenshot of the former is shown in Figure 5.9 on page 129 in §5.3.4.2; one use of the latter is described in the next section.

The implementation of this application requires 265 lines of Scala and 144 lines of JavaScript.

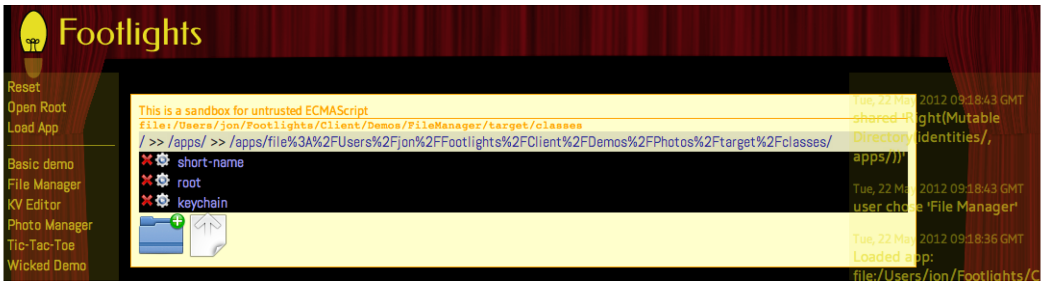
Listing 5.18: Displaying an image: front-end code.

```
context.globals['new_photo'] =
    function new_photo(name, deleteCallback)
{
  /* ... */
  var i = container.appendElement('img');
  i.filename = name;
  i.style['max-height'] = '150px';
  i.style['max-width'] = '150px';
  i.style['vertical-align'] = 'middle';
  /* ... */
}
```

Listing 5.19: Displaying an image: back-end code.

```
override def service(request:WebRequest) =
{
    request.path() match {
      /* ... */
      case OpenAlbum(URLEncoded(name)) =>
        app album name map { album =>
          /* ... */
          setStatus { "Opened album '%s'" format name } ::
          (album.photos map addPhoto toList)
        } fold (
          ex => setStatus("Error: " + ex),
          actions => actions reduce { _ append _ }
        )
      /* ... */
    }
}

private def addPhoto(filename:String) =
  new JavaScript append "context.globals['new_photo']('%s', %s);".
      format(
    filename,
    JavaScript ajax RemoveImage(URLEncoded(filename).encoded)
        asFunction
  )
```



This is a sandbox for untrusted ECMAScript

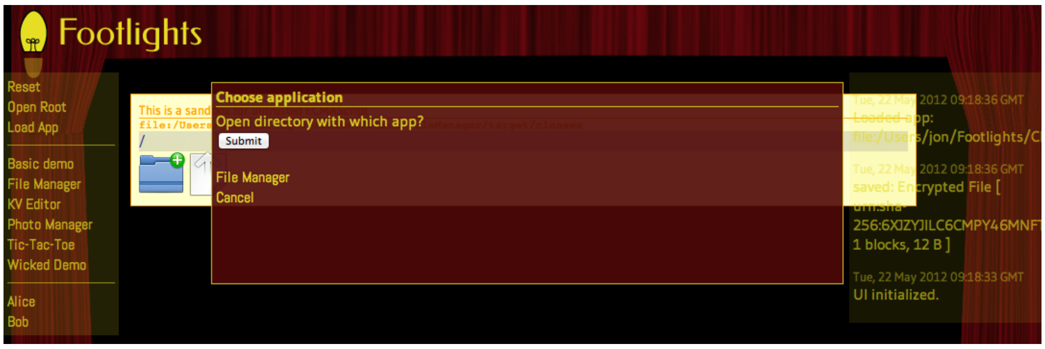
```
file:/Users/jon/Footlights/Client/Demos/FileManager/target/classes
/ >> /apps/ >> /apps/file%3A%2FUsers%2Fjon%2FFootlights%2FClient%2FDemos%2FPhotos%2Ftarget%2Fclasses/
```

```
Tue, 22 May 2012 09:18:43 GMT
shared 'Right(Mutable Directory(identities/, apps/))'

Tue, 22 May 2012 09:18:43 GMT
user chose 'File Manager'

Tue, 22 May 2012 09:18:36 GMT
Loaded app:
file:/Users/jon/Footlights/C
```

Figure 5.13: A basic file manager.



Open directory with which app?

```
Tue, 22 May 2012 09:18:36 GMT
Loaded app:
file:/Users/jon/Footlights/C

Tue, 22 May 2012 09:18:36 GMT
saved: Encrypted File [
urn:sha-
256:6XJZYJILC6CMPY46MNF
1 blocks, 12 B ]

Tue, 22 May 2012 09:18:33 GMT
UI initialized.
```

Figure 5.14: Sharing a directory with the File Manager application.

5.4.5 FILE MANAGER

The File Manager demo, pictured in Figure 5.13, allows the user to manipulate any directory hierarchy that he or she chooses to open.

The File Manager demonstrates how explicit powerbox-driven sharing can be used to delegate partial authority to unprivileged applications. The File Manager itself is an ordinary application with no special privileges: when it starts running, it only has access to its own virtual filesystem. The user can instruct it to modify this private namespace, creating or destroying subdirectories, uploading, downloading or deleting files. By default, no access is granted to other parts of the user's larger filesystem.

The user can, however, choose to open other parts of the filesystem with the File Manager. For instance, Figure 5.14 shows a powerbox triggered by the user sharing an album from the Photos application. If the user chooses the File Manager application from this powerbox, that album's directory will be shared with the File Manager and the user will be able to manipulate it. The user can download or delete photos, upload new ones or open them with another application.

The File Manager demo application is implemented with 258 lines of Scala and 35 lines of JavaScript.

5.5 RELATED WORK

Very little work has been done to date on confining social applications. Section 2.3 discussed numerous proposals to either encrypt content within an existing OSN or to distribute data in a peer-to-peer network. The pure encryption approaches do not alter the application model of their host OSN, and very few of the distributed OSN schemes provide any API for third-party applications.

The confinement problem was first formalised by Lampson in 1973 [153], but his description was based on the prior experiences of both himself [152] and others, such as Schroeder and Saltzer [214]. This problem was: how can an application user be sure that the application is not leaking information? Lampson identified three classes of channels for leaking information: *storage* channels, *legitimate* channels (such as a bill) and *covert* channels, "i.e. those not intended for information transfer at all". The goal of social application confinement is to prevent private user information from leaking through storage or legitimate channels; covert channels are less tractable a problem in commodity systems.

In 1975, Lipner noted that if an application were only provided with a "virtual time" rather than real time, it would even be possible to close covert channels among applications [159]. Unfortunately, Footlights' JVM foundation always permits applications to ask the time.

Various degrees of application confinement has been achieved on many conventional computation platforms. Some examples are given in Section 5.1, but I will highlight three more here.

Stiegler et al.'s Polaris [218] took capability-oriented lessons from the earlier CapDesk project [219] and applied them to a conventional operating system: Microsoft Windows. Polaris sandboxed applications by executing them from a restricted user account and used powerboxes to supply user-driven authority to the unprivileged applications. This allowed unmodified applications to execute with some degree of confinement, subject to Windows' ability to sandbox the applications of restricted users — Watson, Laurie and Kennaway and I have found that Windows lacked some key enforcement primitives [9].

Second, Watson et al. brought capability-oriented sandboxing to FreeBSD through the Capsicum project [1, 9, 17]. This allowed applications to run risky code such as HTML renderers and JavaScript interpreters inside a sandbox with sound OS foundations. This work solidified the boundaries between privileged and unprivileged processes in `sshd` and Chromium and created new separation in `gzip` and `tcpdump`. In all of these cases, Capsicum protections would prevent malicious code from accessing system or user data if it compromised the application's internal logic, e.g. decompression or network packet parsing.

Third, lessons from the CapDesk project and its E programming language have also been applied to Google Caja [351] and ECMAScript 5 [177, 178], both under the influence of CapDesk contributor Mark Miller. Caja, whose implementation is greatly simplified when running on ECMAScript 5, is designed to separate Web scripts from different sources and protect user data from malicious scripts. Footlights' Web UI uses Caja extensively.

Turning from traditional platforms to online social networks, however, there has been much less work done to address the confinement problem.

Baden et al.'s Persona provides its functionality via applications [36], but they are not not confined in the fully-untrusted sense that Footlights uses. In Persona, applications are PHP/MySQL components of a web site. Applications perform cryptography themselves and are not confined by the system. Applications have access to metadata about encryption and are trusted to enforce security policies (e.g. "the Profile application allows only the registered user to write onto the Doc page"). The authors leave as future work an alternate design that splits applications into trusted and untrusted components, running the trusted parts only in the user's browser. When Persona was presented, Footlights was already confining applications, although the API these early apps were given was not as useful or complete as the current one.

Fescher et al.'s "Mr Privacy" allows social applications to be developed without relying on a centralised OSN database [105]. Instead, Mr Privacy uses e-mail to transport information among users. The system is not designed for confinement, however, and content is sent in the clear — the underlying e-mail provider is trusted. A system like Mr Privacy could implement controls around which applications can access what data, but authorisation has not been described yet. The applications themselves run on a standard platform such as a mobile phone or Web browser, so confinement is a task for the platform.

One notable paper amidst the social confinement dearth is Felt and Evans' work on protecting users from malicious social applications [103]. This work studied the behaviour of 150 popular Facebook applications, finding that over 90% had no need of the user data that they were given. The authors also proposed a "privacy by proxy" mechanism inspired by the Facebook Markup Language (FBML) that would allow applications to reference user data indirectly. This is the inspiration for Footlights' placeholders mechanism, described in §5.2.2. Felt and Evans also proposed an encrypted user ID scheme like the one eventually adopted by Facebook in the wake of scenario "Application developers" on page 5.

5.6 SUMMARY OF CONTRIBUTIONS

I have described a platform that demonstrates the viability of distributed social applications. These applications are split into front-end and back-end components, both of which are untrusted and unprivileged, but together they can perform arbitrary computation, manage a UI and interact indirectly with private user data to accomplish user goals via APIs that are familiar to today's application developers.

Application behaviour can be both *confined* and *observed*. Applications have no access to user data by default, but can reference data indirectly without any special permissions. Direct access to user data only occurs with explicit expressions of user intent.

Applications can create information and share it with other users or applications, but this sharing is always under the control of the user. This allows users to be in control of their social sharing without sacrificing the benefits of a generative platform.

6 DISTRIBUTED AUTHENTICATION

It's not what you know, it's who you know.

Workers of Cramps' Shipyard, 1918 [266]

In today's online social networks, integrity is provided by a central party: the Online Social Network (OSN) operator. The operator first authenticates users with a combination of passwords and social graph data, then labels content according to user actions during their authenticated sessions. Users rely on the operator to authenticate other users correctly: a user may talk about "Alice's photo", but a more complete and accurate name for the content would be "a photo that the OSN operator asserts was uploaded by Alice".

In order to move to a decentralised model, indirect certification by a trusted third party must be replaced with direct self-certification. Instead of referencing content as, "what the OSN asserts that Alice said", client software must instead reference "what Alice asserts that she said". I define this to be the *assertion problem*. It is a natural fit for public-key cryptography, where users' client software can make cryptographic assertions that can be verified by anyone.

Public-key cryptography alone does not solve the assertion problem, however. Part of the appeal of OSNs is that they are ubiquitously available: a user can log into a centralised OSN from any computer, anywhere, using a password and some social context which may be implicit or explicit. For instance, logging in from an IP address that has been recently and frequently used by a friend conveys some implicit social context — the user is logging in via a friend's computer or network. Social context can also be established explicitly, by e.g. identifying friends in photographs as described by Kim et al. [142]

Decentralised, public-key–based social networks require an authentication mechanism that is not merely as rigorous as centralised password authentication — trivial for a public-key system — but is also as portable, mobile and usable. As described below, these three requirements preclude traditional approaches to the security of private keys.

This chapter, based on a publication co-authored with Frank Stajano [4], describes a solution to the assertion problem. This solution provides a distributed authentication facility that allows users to retrieve a private key from several *authentication agents*. These agents are software services run by parties who are *honest but curious*: they are assumed to run the authentication protocol correctly, but they are not trusted with confidential plaintext. An offline dictionary attack against this protocol by a malicious authentication agent is infeasible, even if the user's authentication password is very weak e.g. a 4-digit number or dictionary word. Furthermore, the protocol does not allow an attacker to determine if a particular user has registered with an authentication agent, because the authentication agent does not need to know the identity of the user.

6.1 Problems

6.1.1 The assertion problem

As described in Chapters 1, 4 and 5, the goal of the Footlights system is to provide a platform for distributed social applications based on untrusted centralised infrastructure. This alternative social network must provide a means of asserting to a user Alice that a digital artefact was created or authorised by the person she knows as Bob.

Assertions can be made with public-key cryptography, assuming that Bob has ready access to a private key that is kept secret and whose corresponding public key is known to Alice. The problem therefore becomes, "how can Bob access his private key at all times, even when using a friend's computer rather than his own, in order to digitally sign artefacts shared via the OSN?"

In order to solve this overarching problem, the protocol specified in this chapter allows Bob to store his private key with one or more software *authenti-*

cation agents. Bob can authenticate to a set of these agents from any computer, demonstrating knowledge of a secret to re-gain access to his private key.

In order to do this, the protocol must solve several sub-problems. It must allow for the use of weak secrets from any location, require limited trust in authentication agents and provide plausible deniability to users.

6.1.2 Weak secrets

If users could reasonably be expected to memorise cryptographically strong secrets, there would be no need for distributed authentication. This protocol is predicated on the assumption that users cannot memorise cryptographic keys.

The need for this protocol might also be obviated if users always carry a hardware token — such as a mobile phone — that carries cryptographic keys or performs cryptographic operations on the user's behalf. In many cases, however, the user will prefer a friend's large screen to her own small one, so she must either copy her private key to her friend's computer (if the hardware token merely carries keys) or be able to reliably tether her token to her friend's computer (if the token itself performs the relevant cryptography). I claim that these requirements are onerous enough to justify developing an alternative scheme.

For the purposes of this protocol, I require that users must be able to retrieve their private keys without memorising strong secrets such as cryptographic keys or carrying them on hardware tokens.

6.1.3 Location independence

Footlights users must be able to retrieve their keys using any computer.

Users may choose to access Footlights from a shared computer. Authenticating via any computer places some trust in that computer: any authentication protocol that relies on a shared secret is vulnerable to malware. This property is not unique to Footlights. When using a friend's computer, the malware assumption also breaks Facebook's Social Authentication scheme: the malware can learn the user's password and the owner of the computer is one of the "insiders" ideally placed to identify faces of the user's friends via a Man-in-the-Browser (MITB) attack as described by de Barros [85, 128].

When using a computer in a shared computer lab, Internet café, etc., the user would be well-advised not to enter their primary authentication password as a matter of security hygiene. As an alternative, sub-keys could be configured in advance to convey limited authority for a limited time. For instance, before going on vacation, the user might publish a signed notification that "key X may be used to sign messages that contain no URLs until 2 August 2012".

6.1.4 LIMITED TRUST

Users who engage in the distributed authentication protocol should not need to fully trust any participant. Authentication agents should not be able to determine the user's weak secret through offline dictionary attacks. Unless there is widespread collusion, attackers who are also authentication agents should have only marginal advantage over those who are not.

6.1.5 PLAUSIBLE DENIABILITY

Attackers should not be able to determine that a particular user has stored keys with an authentication agent. Similarly, attackers must not be able to determine which users have enlisted the aid of any particular authentication agent.

6.2 PRINCIPALS

Bob, B, is a user of a distributed online social network such as Footlights or PeerSoN [65]. This network has no central, trusted authority to certify Bob's digital identity, so his client software proves who he is using Bob's private key.

Wishing to access the network from abroad, Bob prepares a temporary private key K_B^{-1} which he advertises to his peers along with a set of constraints. For instance, Bob could prepare one key per calendar week and declare them valid for sharing photos but not publishing applications. This would ensure that, if a key is compromised by an untrustworthy computer, the damage that can be done is limited. He then enlists software principals Alice, Alexa, Alicia, etc. $(A_0, A_1, A_2, \dots)$ to act as *authentication agents* on his behalf: each will store some portion of the private key or keys, which will only be given out to Bob later if he authenticates himself with the weak password k_B.

I assume that Bob is able to recover his public key K_B and those of his agents

$K_{A_0}, K_{A_1}, \ldots$ from a public but untrusted source such as a key server. As an untrusted service, it provides keys for confidentiality of communications, not authentication of principals. The only secret used for authentication is k_B.

6.2.1 INSIDERS

I assume that Alice, Alexa, etc. are honest-but-curious according to the definition of "Honest but curious" on page 16. I also assume that insiders — or the malware they are infected with — will not collude *en masse*, following the example of Beimel and Chor [38]. The distributed authentication protocol will make it difficult for insiders to identify each other and so collude. Nonetheless, in order to minimise the chance of collusion, Bob might choose agents who are in different social cliques or are business competitors.

A malicious agent may attempt to impersonate Bob to other agents. In this situation, an insider acts as the outsider described below, but with the advantage of the information that it is storing for Bob.

6.2.2 OUTSIDERS

I assume that there is a malicious outsider, Mallory. His goal may be to learn Bob's private key or simply to map his usage of the system by determining which agents — if any — are storing key material for Bob.

Mallory is modeled as the powerful Dolev–Yao attacker [92]. He can eavesdrop on communication between Bob and his authentication agents, so the content of these communications must be protected. He can initiate communication with any agent, so he can attempt to impersonate Bob to an authentication agent. He can also intercept messages, so he can attempt to impersonate an authentication agent to Bob.

The key limitation on the outsider is the dictionary attacks he performs against Bob's password must be *online*. When Mallory impersonates Bob to an authentication agent, the agent will impose a mandatory timeout between each authentication attempt. This timeout may be static or a time-varying function, such as an exponential back-off from a fixed initial timeout, but authentication agents impose it to keep outsiders from becoming *de facto* insiders.

As a matter of practicality, a blind exponential timeout provides an attacker with an opportunity to perform a Denial of Service (DoS) attack: by repeatedly guessing Bob's password and driving up the timeout, Bob himself might be prevented from logging in. This threat might be mitigated by intrusion detection mechanisms, proposed by Denning [87], which probabilistically detect or block attacks. It might also be mitigated by using secondary communication channels, analogous to the *fallback authentication* described by Just and Aspinall [138, 139] or Schechter et al. [212, 213], to restore Bob's access during attacks. These mechanisms could be used to combat denial of service attacks and enhance availability, but they are not trusted to provide confidentiality or integrity properties.

6.3 PROTOCOLS

I present several security protocols, beginning with a straw-man that relies on a trusted third party and building towards a protocol that addresses all of the problems in Section 6.1. As introduced above, I use the following notation:

k_B	Bob's weak secret (password)
K_x	A public key associated with the name x
K_x^{-1}	The private key corresponding to public key K_x
K_t	A temporary public key, generated for use in session set-up
$\{M\}_{K_x}$	A message M encrypted under public key K_x with NM-CPA [41]

Non-malleability of the encryption mechanism is required to prevent an active attacker from creating new authentication tokens from old ones. These tokens consist of information derived from a secret, concatenated with a nonce to prove freshness; an ability to construct related tokens could allow challenges to be re-used with new nonces.

None of the protocols below use explicit integrity mechanisms such as digital signatures. In particular, the veracity of K_A is not known to the supplicant, so signing information with K_A would not yield useful integrity guarantees. Instead, supplicants assume that messages encrypted to K_t have been generated by a principal to whom K_t has been given.

6.3.1 TRUSTED THIRD PARTY

The first protocol is trivial: Bob stores his private key in plaintext on a trusted server. After retrieving Alice's public key from the public key server introduced above, the protocol is simply:

$$
\begin{aligned}
B \rightarrow A &: \{B, K_t\}_{K_A} && (6.1)\\
A \rightarrow B &: \{n\}_{K_t} && (6.2)\\
B \rightarrow A &: \{k_B, n\}_{K_A} && (6.3)\\
A \rightarrow B &: \left\{K_B^{-1}\right\}_{K_t} && (6.4)
\end{aligned}
$$

in which k_B is Bob's weak authentication secret, K_t is a temporary key used by Bob to provide confidentiality until he recovers his private key K_B^{-1} and n is a nonce selected by Alice, which prevents Mallory from performing a replay attack should he later learn the value of K_t.

This protocol is analogous to a common practice in fallback authentication: if Bob loses his password to a Web service, he can have a reset token sent to his e-mail account. The e-mail provider is trusted implicitly.

The protocol bears a resemblance to the Kerberos protocol described by Newman et al. [192, 264]: a supplicant (Bob) proves his identity to a trusted system (Alice), which releases a token (Bob's private key) that can be used to obtain other services (interaction with other users). It is different from Kerberos, however, in that the authentication agent is not a source of authority: it does not generate or certify keys, it merely stores them.

Attacks Eavesdropping attacks, as defended against in the Kerboros realm by Bellovin and Merritt's original Encrypted Key Exchange (EKE) protocol [43], do not apply because the message $\{B, K_t\}$ which Bob sends to Alice is entropy-rich. The presence of K_t confounds offline dictionary attack by eavesdroppers.

The protocol also prevents active outsiders (§6.2.2) from conducting successful online dictionary attacks, since Alice can limit the rate of incoming password guesses. Because of the rate-limiting of Mallory's online dictionary attack, Bob's password can be as weak as an English word.

There are, however, two very obvious attacks against the system. First, insider Alice can simply read Bob's private key and password in the clear. Second, outsider Mallory can impersonate Alice in order to learn Bob's password in message (6.3). Bob can download a K_A from the public-but-untrusted key server which is attributed to Alice but, without a shared secret or trusted authority, he has no way to verify that it is actually Alice's key.

6.3.2 SEMI-TRUSTED STORAGE

A slight improvement on the Trusted third party scheme is the Semi-trusted storage scheme. In this scheme, Alice does not hold Bob's password and private key in the clear. Instead, she holds a cryptographic hash[1] of the password $h(k_B)$ and Bob's private key encrypted using his weak password k_B, $\left\{K_B^{-1}\right\}_{k_B}$. The protocol is very similar to that of the Trusted third party scheme:

$$\begin{aligned}
B \rightarrow A &: \{B, K_t\}_{K_A} && (6.5)\\
A \rightarrow B &: \{n\}_{K_t} && (6.6)\\
B \rightarrow A &: \{h(k_B), n\}_{K_A} && (6.7)\\
A \rightarrow B &: \left\{\left\{K_B^{-1}\right\}_{k_B}\right\}_{K_t} && (6.8)
\end{aligned}$$

After receiving message (6.7), Alice verifies that $h(k_B)$ matches her stored copy, then sends Bob $\left\{K_B^{-1}\right\}_{k_B}$ in message (6.8).

Attacks This protocol prevents a truly disinterested Alice from reading Bob's secrets, but it does little to stop a curious Alice or a clever Mallory.

Alice can clearly mount an offline dictionary attack against the stored password — the number of iterations in the password hash affects only the cost of the attack, not the ability to do it. Since the password is assumed to be weak, an offline dictionary attack should be expected to succeed with high probability.

[1]As a password hash, it would be salted and iterated; the details are left to implementation.

Furthermore, if Mallory impersonates Alice, Bob will send him the same hash that Alice stores — this occurs in message (6.7). Based on this hash, Mallory can mount the same offline dictionary attack as Alice.

6.3.3 SECRET SHARING

A logical and straightforward extension to this protocol is for Bob to spread his private key across several agents Alice, Alexa, Alicia, etc. using a k of n secret sharing scheme[2] as proposed by Shamir [216]. In this case, Alice stores $h\left(K_{A_i}|k_B\right)$, which is a hash of k_B personalised to her. Instead of the private key K_B^{-1} she stores D_i, which is one portion of the key K_B^{-1} shared according to the secret sharing scheme. The protocol between Bob and an agent is now:

$$\begin{aligned} B \rightarrow A_i &: \{B, K_t\}_{K_{A_i}} && (6.9) \\ A_i \rightarrow B &: \{n\}_{K_t} && (6.10) \\ B \rightarrow A_i &: \{h\left(K_{A_i}|k_B\right), n\}_{K_{A_i}} && (6.11) \\ A_i \rightarrow B &: \{D_i\}_{K_t} && (6.12) \end{aligned}$$

Attacks This addition to the protocol prevents Alice, Alexa, etc, from reading Bob's private key. If the secret sharing scheme provides semantic security, a malicious agent will not be able to use it in an offline dictionary attack. However, there is nothing to prevent her or impostor Mallory from attacking the weak password by brute force and, once successful, impersonating Bob to other key recovery agents.

This distinction does not change the parameters of the model — it still fits the definition of "Honest but curious" on page 16 — but there may be a social value in the difference. It requires a different degree of brashness for Alice to impersonate Bob than for her to peek at a value that Bob has asked her to keep secret. As a technical difference, however, it does not limit the trust which Bob must place in his authentication agents; problem "Limited trust" on page 144 is not solved by this protocol.

[2]For the purposes of describing the high-level protocol, no specific scheme is specified.

6.3.4 COLLISION-RICH PASSWORD HASHING

This protocol can be further improved by using collision-rich hash functions as previously used by Lomas and Christianson [162] (see Section 6.6, "Related work", for further details). These functions can be built by discarding some of the output of cryptographic hash functions, as in equation (6.13):

$$h_M\left(x\right) = h\left(x\right) \bmod M\,. \tag{6.13}$$

This modified hash function introduces collisions as long as $M \ll 2^{N/2}$, where N is the output size of hash function h.

Collision-rich hash functions frustrate dictionary attacks because they enlarge the equivalence classes of inputs that map to the same output. If a large number of salted passwords map to the same output hashes, then an offline dictionary attack against a hash will only discover the equivalence class of possible inputs. Checking these inputs requires an online consultation of some other password oracle.

For the purposes of this protocol, I define the collision-rich hash function $h_{M,i}$ for agent A_i according to equation (6.14), where $h\left(x\right)$ is a cryptographic hash function:

$$h_{M,i}\left(k_B\right) = h_M\left(K_{A_i}|k_B\right) = h\left(K_{A_i}|k_B\right) \bmod M\,. \tag{6.14}$$

The protocol now becomes:

$$\begin{aligned}
B \rightarrow A_i &: \{B, K_t\}_{K_{A_i}} && (6.15)\\
A_i \rightarrow B &: \{n, M\}_{K_t} && (6.16)\\
B \rightarrow A_i &: \{h_{M,i}\left(k_B\right), n\}_{K_{A_i}} && (6.17)\\
A_i \rightarrow B &: \begin{cases} \{D_i\}_{K_t} & \text{if } h_{M,i}\left(k_B\right) \text{ correct} \\ \{\text{random}\}_{K_t} & \text{otherwise} \end{cases} && (6.18)
\end{aligned}$$

where M is a number chosen by Bob when he enlists Alice as a key recovery agent. This technique reduces the trust that must be placed in authentication agents, satisfying problem "Limited trust" on page 144. The degree to which this trust is reduced is governed by M, which balances attacks by insiders against attacks by outsiders. As M increases, it becomes more difficult for an attacker to randomly guess a $h_{M,i}(k_B)$ that satisfies Alice. This is the Outsider dictionary attack in §6.4.3. On the other hand, the larger the value of M, the smaller the equivalence class of k_B. This facilitates the Insider dictionary attack in §6.4.4.

6.3.4.1 LARGE-M ATTACK

Alice stores the value $h_{M,i}(k_B)$, where M is decided by Bob when he recruits Alice as an authentication agent. As $\log_2 M$ increases, h_M has fewer collisions. If Mallory impersonates Alice, he could send a large value of M to Bob in equation (6.16). If Bob were to reply with the message $h(K_M|k_B)$, Mallory would be able to conduct an offline dictionary attack and learn k_B.

Rather than responding with a collision-poor hash, Bob should ignore some values of M. The test might be user-driven: the value of M could be displayed to Bob or used to influence his UI in some recognisable way. If Bob observes a different value of M from what he usually sees, he can cancel the login attempt. It could also be automated, rejecting values of M that lie outside of some reasonable range related to the strength of his password. Finally, all authentication agents could be required to use the same value of M. In that case, Mallory would need to impersonate all of Bob's authentication agents in order to convince him of a false value of M. Otherwise, Bob's client software could observe the partitioning of M values and abort the authentication attempt.

6.3.4.2 IMPOSTOR IDENTITY DISCLOSURE ATTACK

If Mallory impersonates Alice, he will not be able to learn Bob's password easily for the reasons given in §6.4.4, "Insider dictionary attack". Neither will he be able to perform a completely successful middleperson attack, since the hash $h(k_B|K_A)$ is bound to Alice's public key. He would, however, learn that Bob has stored his private key with Alice by virtue of the fact that Bob has attempted authentication. This information is probabilistic in nature: since Bob has not authenticated, Mallory cannot be sure that it was Bob authenticating rather than some impersonator. If such impersonations occur regularly, however, Bob's plausible deniability is limited.

6.3.5 COLLISION-RICH IDENTITY HASHING

In order to prevent identity disclosure in the face of the Impostor identity disclosure attack (§6.3.4.2), one more layer of complexity can be added: collision-rich hashing of Bob's identity. To prevent these hashes from acting as *de facto* persistent identifiers, Bob should send each agent a collision-rich hash of his identity and the agent's when registering, using a very low modulus of his choice. If the agent cannot disambiguate him from other users that she provides the authentication service to, he can try again using a different modulus.

$$B \rightarrow A_i \quad : \quad \{h\left(A_i|B\right) \bmod N, N, K_t\}_{K_{A_i}} \tag{6.19}$$

$$A_i \rightarrow B \quad : \quad \text{try again} \tag{6.20}$$

$$B \rightarrow A_i \quad : \quad \{h\left(A_i|B\right) \bmod N', N', K_t\}_{K_{A_i}} \tag{6.21}$$

$$A_i \rightarrow B \quad : \quad \{n, M\}_{K_t} \tag{6.22}$$

$$B \rightarrow A_i \quad : \quad \{h_{M,i}\left(k_B\right), n\}_{K_{A_i}} \tag{6.23}$$

$$A_i \rightarrow B \quad : \quad : \begin{cases} \{D_i\}_{K_t} & \text{if } h_{M,i}\left(k_B\right) \text{ correct} \\ \{\text{random}\}_{K_t} & \text{otherwise} \end{cases} \tag{6.24}$$

This protocol gives the same probability of successful dictionary attack by insider or outsider, and it also counters the Impostor identity disclosure attack for the same reasons. It does not mitigate the potential for traffic analysis: a technically competent adversary could observe that Bob has connected to Alice. However, I consider traffic analysis to be beyond the scope of this distributed authentication protocol, except to say that the protocol is not incompatible with anonymity technologies such as Tor [91].

6.4 PASSWORDS AND PROBABILITIES

In this section, I evaluate the probability of an attacker learning Bob's password. This attacker may be an insider — a curious authentication agent — or an outsider. Before evaluating these probabilities, I must first provide a more

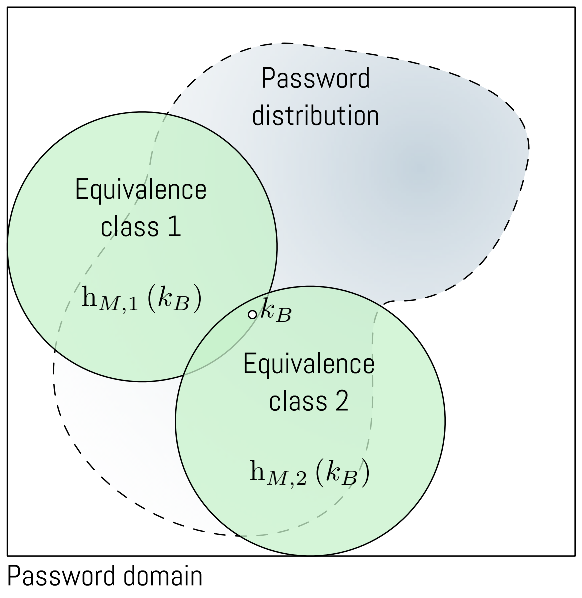


Figure 6.1: Password domain, distribution and equivalence classes.

rigorous definition of "weak password".

The security of this distributed authentication protocol must not rely on Bob memorising a strong password (see problem "Weak secrets" on page 143). Instead, Bob's password will either be a randomly-assigned word from a small dictionary or a user-chosen password from a known password distribution.

Bob's password is drawn from a password distribution which is assumed to be known to the attacker. This distribution is a function over a domain of possible passwords, as shown in Figure 6.1. The distribution shown in this figure does not cover all possible passwords: some passwords in the domain have a probability of zero.

If Alice's identity is A_i, she holds the value $h_{M,i}(k_B)$. I assume that she is able to perform an offline dictionary attack in order to determine a set or *equivalence class* of passwords that hash to the same $h_{M,i}(k_B)$ as k_B. Under the random oracle assumption, the size of this equivalence class of passwords will be Σ/M on average, where Σ is the size of the password domain. If Alice can learn other authentication agents' equivalence classes, she might be able to find a small intersection among them, as shown in Figure 6.1, revealing Bob's password with

Listing 6.1: Random words from a 10,000-word English dictionary.

```
aside      crypt      curd       driest     finite
funding    morale     nimbler    oven       pasta
```

Listing 6.2: Filtering words from a SCOWL word list.

```
./mk-list english 35 \              # SCOWL list, recommended size
    | grep "^[a-z]\{4,7\}$" \       # keep 4-7 letter words
    | grep -v "\(ed\)\|\(s\)$" \    # drop "pained", "pains"...
    | head -n10000                  # keep exactly 10,000 words
```

high probability.

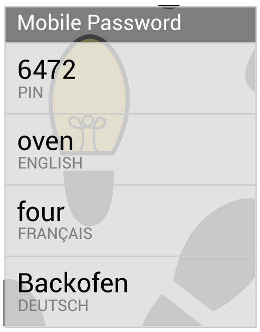
I will now consider two realistic distributions that Bob's password might be drawn from: a discrete uniform distribution over a small domain and a user-chosen distribution over a large domain.

6.4.1 UNIFORM DISTRIBUTION

If Bob's password is drawn from a uniformly random distribution, the size of the password distribution will necessarily be equal to the size of the password domain Σ. k_B must be drawn from a small dictionary in order to aid memorability: the protocol will not require Bob to memorise random strings such as `"NBu6#Z4NI:"`, a feat whose difficulty has previously been demonstrated in password research by Yan et al. [238, 239]

Listing 6.1 shows a set of ten words randomly chosen from an English dictionary of 10,000 common words. This dictionary was created by filtering a longer list from Atkinson's SCOWL (Spell Checker Oriented Word Lists) project [339], which is the basis for the GNU Aspell spelling checker. My filtering, shown in Listing 6.2, restricts the dictionary to words that are 4-7 letters long and contain only unaccented Latin letters. It also excludes some very similar word pairings such as plural versions of words and past participles (e.g. "pain" and "pained").

The resulting words are simple and frequently-used. Some of the psychology literature [45, 64, 129] suggests that low-frequency words may be easier to



Mobile Password

6472	PIN
oven	ENGLISH
four	FRANÇAIS
Backofen	DEUTSCH

Figure 6.2: Possible presentation of a random word from a small dictionary.

recognise than high-frequency words, but this protocol depends on *recall*: bringing a particular word out of long-term memory, not recognising it in a list. For such a task, I have filtered for high-frequency words, but low-frequency words could be filtered for just as easily. Empirical evaluations of the experimental psychology of recall are beyond the scope of this work.

Whether high- or low-frequency words are easiest to recall, the underlying mechanism is the same: a random number is chosen from the range $[0, 9999]$ and used as an index into the dictionary. Several dictionaries could be used concurrently, as shown in Figure 6.2, possibly increasing memorability: the user need only remember one of several equivalent words.

6.4.2 NON-UNIFORM DISTRIBUTION

The other possible source of a weak secret is a user-chosen password. The empirical evaluation of user-chosen passwords has recently been given new life through studies of real-life password data, e.g. Dell'Amico et al.'s recent work on password cracking [86] and especially the RockYou data set, a collection of 32 M user passwords leaked in 2010 and previously studied by Weir et al. [229] and Bonneau [54]. I have used this data set to illustrate the effect of collision-rich hashing on user-chosen password distributions.

Figure 6.3 on the following page shows the frequency distribution of passwords in the RockYou corpus. The comparison to a power-law distribution is for visual reference only: this reference is not an empirically-fitted distribu-

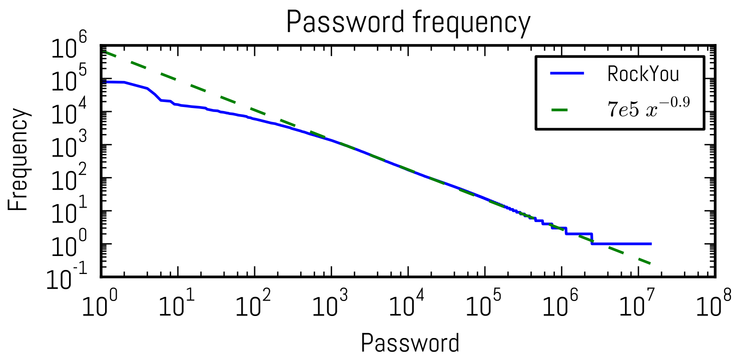


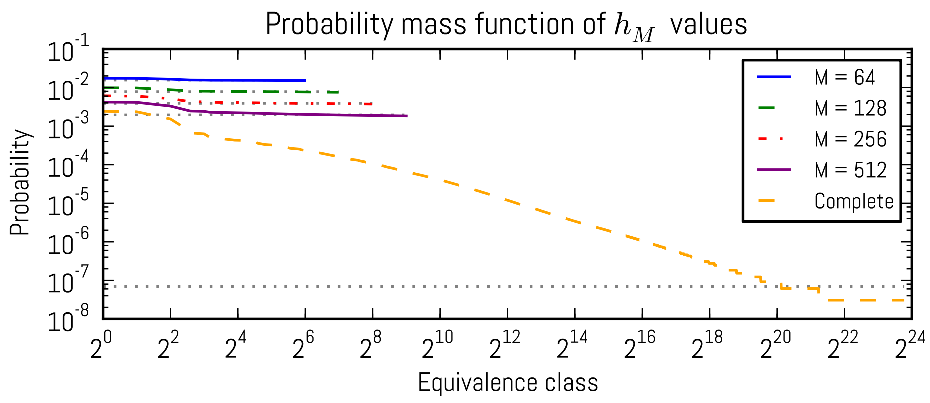
Figure 6.3: Frequency of RockYou passwords v. a power law distribution.

tion. The plot illustrates a large variation in password frequencies, which means that an attacker employing an optimal guessing strategy will require much less work than bruce-force search to learn a password with probability $\alpha < 1$ (see Bonneau's G_α metric [54]). The effect of a collision-rich hash on this distribution is to redistribute unlikely passwords into the same password equivalence classes as likely ones, flattening the probability mass function (pmf).

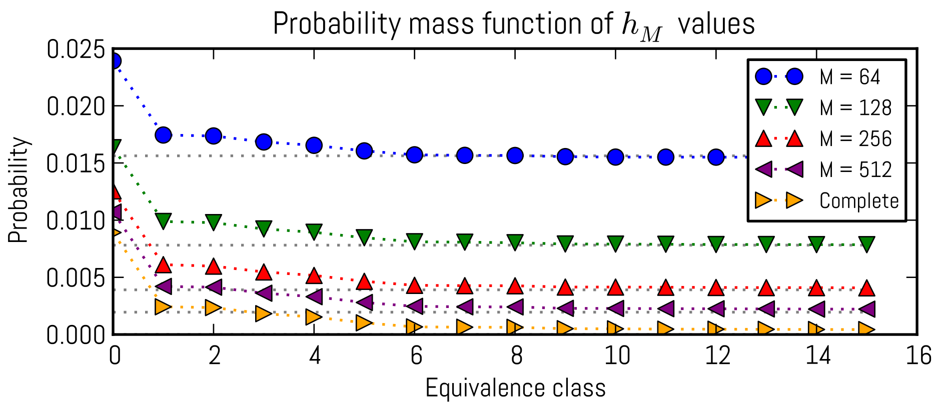
Figure 6.4a on the next page illustrates the effect of M on the distribution of h_M values. The complete password distribution shows a wide variation in probabilities: approximately five orders of magnitude. The distribution of corresponding collision-rich hashes, however, is greatly flattened: the smaller the value of M, the more even the distribution of equivalence classes.

Figure 6.4b on the facing page shows a more detailed view of the most probable h_M values. If x is a randomly-chosen password from the RockYou corpus and X_i is the i^{th} most common password, then $\Pr\left[h_{64}(x) = X_0\right]$ is 53.6% greater than $1/64$ (the probability of a uniformly random password with $n = 64$) and $\Pr\left[h_{64}(x) = X_1\right]$ is only 11.9% greater than $1/64$. In the original distribution, the most-frequent password occurred orders of magnitude more often than the least-frequent. In the collision-rich distribution with $M = 64$, the spread between the most- and least-likely equivalence classes is less than a factor of two.

The use of a collision-rich hash function to create equivalence classes among passwords yields a probability distribution that reduces the advantage an optimal attacker has over brute-force guessing.



(a) Probability mass function (logarithmic).



(b) Detail of most likely h_M values.

Figure 6.4: Probability distribution of h_M values for RockYou passwords.

In Section 6.3, I developed an authentication protocol that uses collision-rich password hashes. Having introduced two password dictionaries — one with 10,000 equally-likely words and one chosen by real users — I now turn to the evaluation of attack success probability using this authentication protocol.

6.4.3 OUTSIDER DICTIONARY ATTACK

In order for an outsider, Mallory, to obtain Bob's credentials, he must convince at least k authentication agents to reveal their true secret shares.

As described in §6.2.2, I assume that each authentication agent imposes a timeout between guesses that may itself be a function of time, e.g. exponential back-off. Under this assumption, time can be viewed as a discrete quantity: the maximum number of guesses that an attacker can have submitted to each authentication agent. I assume the attacker sends guesses in parallel to all agents.

6.4.3.1 UNIFORM PASSWORD DISTRIBUTION

If Bob's password is chosen from a uniformly random distribution, attacker Mallory has no information about which passwords he should attempt to guess first. Furthermore, assuming that each portion of the shared secret is indistinguishable from random, Mallory is unable to learn partial information from any single authentication agent: the only way to determine that he has guessed the correct values of $h_{M,i}\left(k_B\right)$ is to successfully re-assemble Bob's private key. Thus, the optimal guessing strategy is to send random guesses of $h_{M,i}\left(k_B\right)$ to each authentication agent until k genuine secret shares have been recovered — a brute force attack.

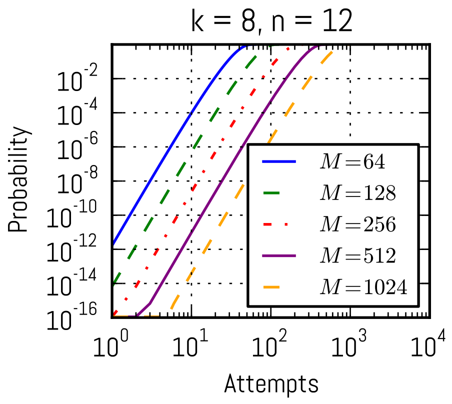
Since Mallory has no information about Bob's password, each of the M guesses that he might send to an authentication agent is equally likely to be correct: the probability is $\frac{1}{M}$. Thus, the cumulative distribution function (CDF) for any particular agent having revealed her genuine secret share at time t is simply $\frac{t}{M}$. Mallory's interaction with each authentication agent is independent of his other interactions, so each can be considered a Bernoulli trial. The probability that Mallory will have assembled k shares of the secret from n authentication agents is thus given by the binomial CDF in equation (6.28).

$$\begin{aligned}
p_r\left(k,n,t,M\right) &= \Pr\left(X \geq k\right)\big|_{p=\frac{t}{M}} & (6.25)\\
&= 1-\Pr\left(X \leq k-1\right)\big|_{p=\frac{t}{M}} & (6.26)\\
&= 1-\sum_{i=0}^{k-1}\binom{n}{i}p^{i}\left(1-p\right)^{n-i}\big|_{p=\frac{t}{M}} & (6.27)\\
&= 1-\sum_{i=0}^{k-1}\binom{n}{i}\left(\frac{t}{M}\right)^{i}\left(\frac{M-t}{M}\right)^{n-i}. & (6.28)
\end{aligned}$$

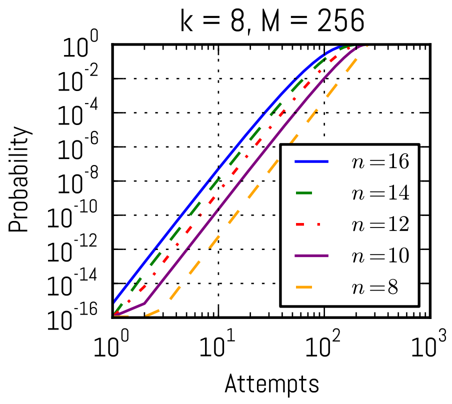
The effect of varying M, k and n is shown in Figure 6.5 on the next page. Varying M or n creates a shift on the logarithmic success plots: if M is doubled, the attacker's success probability will be reduced by a factor of ~100 for a given number of authentication attempts. A similar shift is observed when n is halved. The more interesting variation, shown in Figures 6.5c, 6.5d and 6.5e, is that of varying k. Unlike M and n, which cause a shift on the log-log attack success plot, a linear variation in k changes the slope of the attack success plot. The effect of varying k overwhelms that of other parameters, such as n: it is more difficult for an attacker to randomly authenticate 10 times out of 50 (Figure 6.5e) than 5 out of 5 (Figure 6.5c) or 6 out of 12 (Figure 6.5d).

Using these graphs, Bob can choose a desired level of security as the pair (p_r, t): a desired maximum probability of attacker success after a certain number of time intervals. This pair represents a point on the graphs in Figure 6.5 on the following page, which can be used to select values of M, k and n.

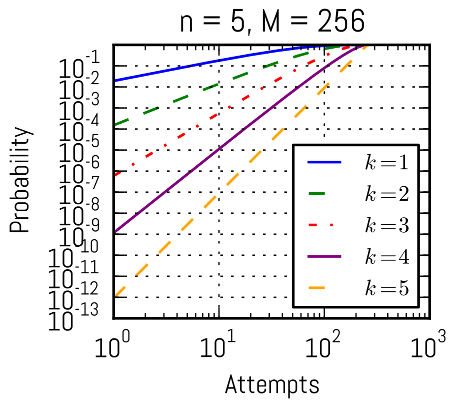
Suppose Bob registers 12 authentication agents and distributes an 8-of-12 secret share to them. An outsider performing an online dictionary attack can expect to achieve a success probability $\alpha = 10^{-4}$ (equivalent to guessing a random 4-digit PIN with one attempt) after ~10 parallel online authentication attempts if $M = 64$. If $M = 256$, it will take ~50 attempts to reach this probability of success, but if $M = 1024$, it will take ~2,000 attempts. If each authentication attempt triggers an exponential back-off, more than ~20 attempts may be prohibitive: a five-second timeout that doubles on each failure will consume 60 days of real time. In this scenario, 50 attempts would require ~35 million years.



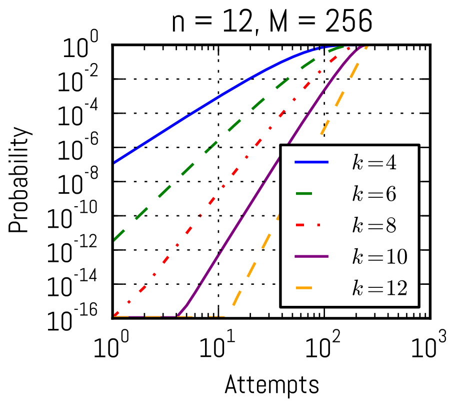
(a) Effect of varying M.



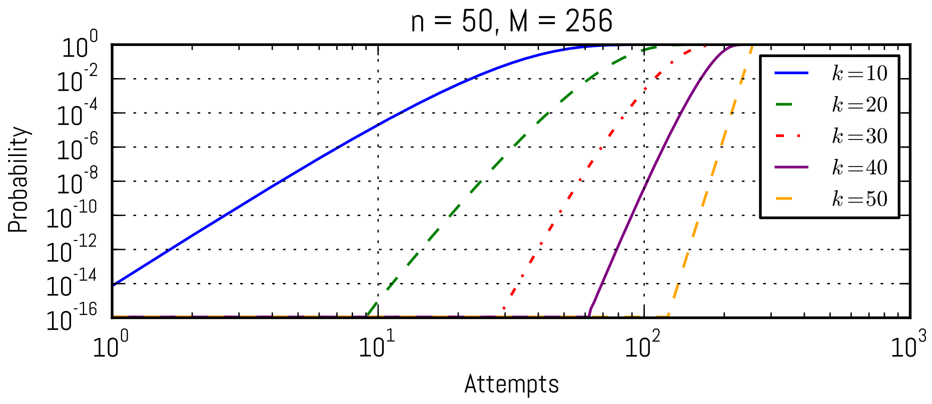
(b) Effect of varying n.



(c) Effect of varying k (small N).



(d) Effect of varying k (medium N).



(e) Effect of varying k (large N).

Figure 6.5: Probability of successfully attacking a uniform distribution.

6.4.3.2 NON-UNIFORM PASSWORD DISTRIBUTION

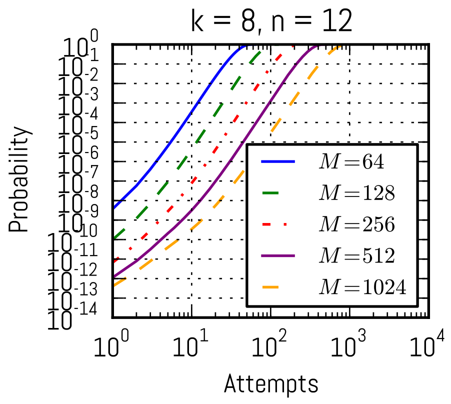
If Bob's password is chosen from a non-uniform distribution, it is easier for the attacker to guess, but M, n and k can be chosen to frustrate practical attacks.

In this case, the attacker still proceeds by sending likely h_M values to each authentication agent and attempting to combine k of n responses into a valid public key. Unlike the uniformly random case, however, the cumulative distribution function is non-linear. Rather than the simple ratio t/M, where t is the discrete time, the CDF is given by actual password frequencies in the distribution that Bob's password is drawn from.

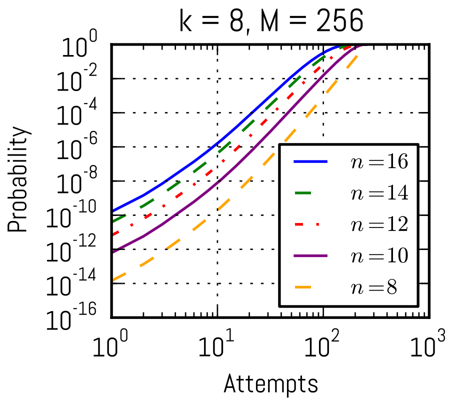
I use the CDF of the RockYou corpus with collision-rich hashing as the probability that any independent authentication agent will have revealed her true secret share to the attacker at time t. This data-driven approach does not yield a closed-form solution for the attacker's combined CDF as in equation (6.28), but it is still a binomial distribution as shown in equation (6.31).

$$\begin{aligned} p_r(k,n,t,M) &= \Pr(X \geq k) && (6.29) \\ &= 1 - \Pr(X \leq k-1) && (6.30) \\ &= 1 - \sum_{i=0}^{k-1} \binom{n}{i} p^i (1-p)^{n-i} \,\big|_{p=\mathrm{CDF}_M(t)} . && (6.31) \end{aligned}$$

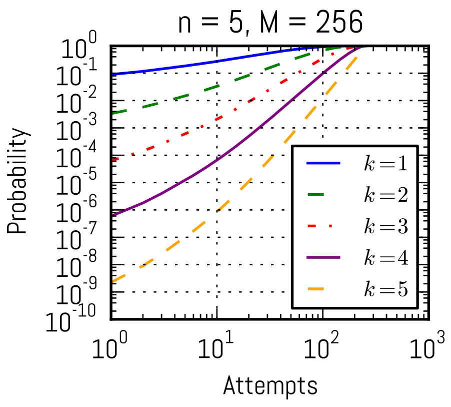
Figure 6.6 on the next page shows the attack success probabilities in this scenario. All of these plots show higher success probabilities than the uniformly random case, but the effect is most pronounced in the $k = 10$ line in Figure 6.6e. Here, the minimum attack success probability of approximately 10^{-8} is much higher than the corresponding value of 10^{-16} in Figure 6.5e. Nonetheless, practical values of M, n and k can be chosen (e.g. $k = 8$, $n = 16$, $M = 256$) that keep the attacker's success probability below 10^{-4} until time $t = 20$, which §6.4.3 showed might correspond to 60 days of real attacker time.



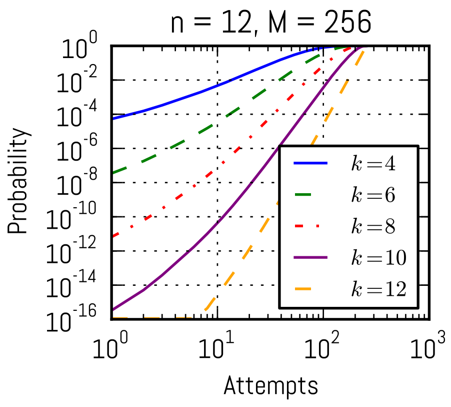
(a) Effect of varying M.



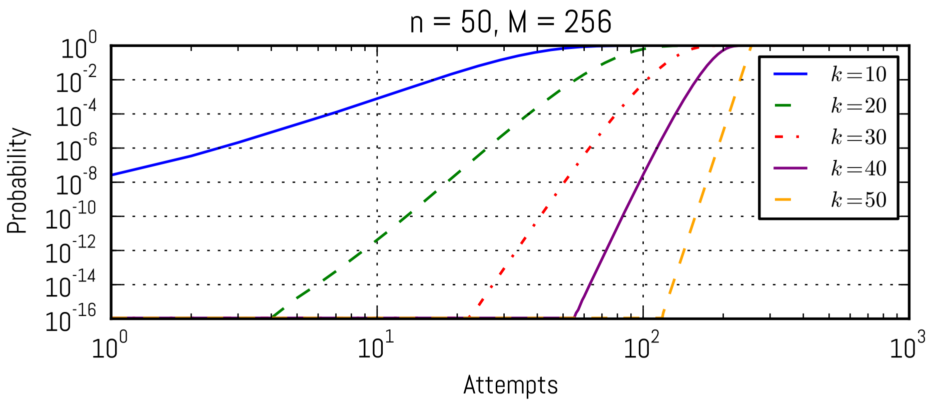
(b) Effect of varying n.



(c) Effect of varying k (small N).



(d) Effect of varying k (medium N).



(e) Effect of varying k (large N).

Figure 6.6: Probability of successfully attacking the RockYou distribution.

6.4.4 INSIDER DICTIONARY ATTACK

I will now evaluate the probability that an insider Alice or a successful middleperson Mallory (see §6.2.2) will be able to exploit the collision-rich hash of Bob's password with Alice's public key in order to learn Bob's password.

If Alice performs an offline dictionary attack against $h_{M,i}(k_B)$, she will learn the equivalence class $[k_B]$ for which equation (6.32) holds:

$$h_{M,i}(x) = h_{M,i}(k_B)\,\forall x \in [k_B] \ . \tag{6.32}$$

Under the random oracle assumption, this equivalence class will be a uniformly random sample of the original password distribution whose size, on average, will be $|X|/M$ where $|X|$ is the size of the password domain X. Alice can test passwords in this equivalence class according the optimal ordering given by the original password distribution; she simply needs to impersonate Bob to at least k other authentication agents, requesting secret shares and checking to see if they combine to form Bob's private key.

As stated in §6.2.1, I assume that the authentication agents do not collude, according to precident in the literature. In that section I also suggest measures to reduce the likelihood of such collusion, making the assumption plausible.

6.4.4.1 UNIFORM PASSWORD DISTRIBUTION

If Bob's password is drawn from the uniform distribution, then each of his possible passwords are equally likely. If $\frac{|X|}{M} \leq M$, Alice can try each of the $|X|/M$ possible password classes against the other $n-1$ authentication agents. If $n-1 \geq 2k$, she can even query multiple sets of agents in parallel. The probability that Alice will successfully learn Bob's password is, therefore:

$$p(t) = t\frac{M}{|X|} \cdot \left\lfloor \frac{n-1}{k} \right\rfloor \ . \tag{6.33}$$

This is a linear function of the discrete time t, governed by the number of possible passwords ($|X|/M$) and the number of parallel queries which can be executed, $\left\lfloor \frac{n-1}{k} \right\rfloor$. For instance, if $M = 128$, $|X| = 10^4$, $n = 30$ and $k = 10$, then $p(t) = 2.56 \times 10^{-2}\,t$. If $|X| = 10^6$, however, corresponding to a 6-digit PIN, then $p(t) = 1.28 \times 10^{-4}\,t$ as long as $k > \frac{n-1}{2}$. After 15 discrete-time guess attempts (two days of exponential back-off from 5 s), the probability that insider Alice will have guessed Bob's password is 1.92×10^{-3}.

If $k > \frac{n-1}{2}$, the tunable parameters available to Bob are M and $|X|$. Increasing the password space obviously decreases the attacker's probability of success, but so does reducing the value of M. This is a fundamental trade-off: smaller values of M increase the difficulty of the Insider dictionary attack and decrease the difficulty of the Outsider dictionary attack.

6.4.4.2 Non-uniform password distribution

In the case where Bob chooses his own password from a distribution like the RockYou distribution, the Insider dictionary attack's success probability is almost identical to that of the Outsider dictionary attack.

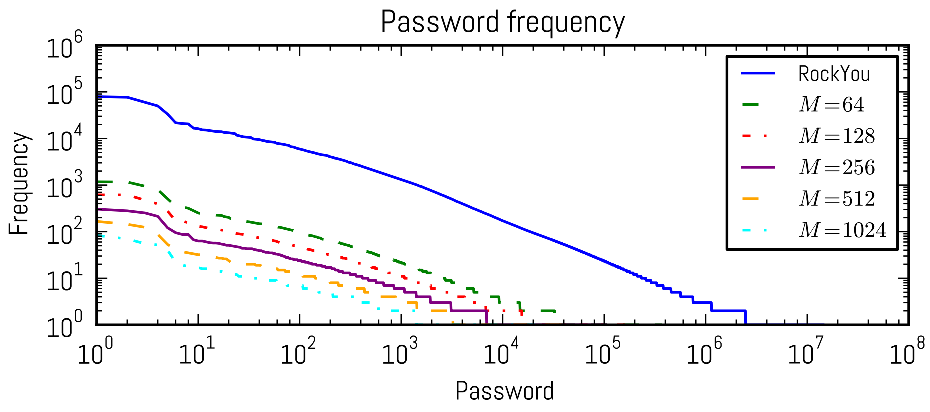
If Alice (or middleperson Mallory) learns $h_M(k_B)$, she can perform an offline dictionary attack in order to learn a set of $|X|/M$ passwords that includes Bob's password. Unlike the case of the uniform password distribution above, however, the overall size of the password space means that narrowing the problem down to $|X|/M$ possibilities does not give the attacker much advantage: $|X|/M$ is still much larger than M.

Figure 6.7 on the facing page shows the RockYou password distribution compared with several sampled distributions drawn from it using different several values of M. The overall shapes of the sampled curves in Figure 6.7a on the next page are largely the same as the original distribution. More importantly, the number passwords with frequency $f > 1$ is larger than M, so Figure 6.7b on the facing page looks very similar to its analogue in Figure 6.4 on page 157.

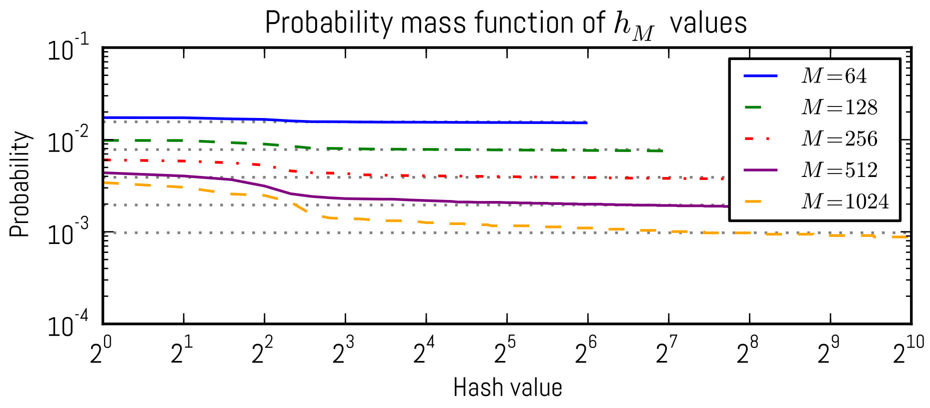
The PMF of the h_M values for sampled RockYou distributions are visually similar to those of the original distribution, owing to the vast size of the original distribution. This leads to a Figure 6.8 on page 166 similar to Figure 6.6 on page 162: the insider has little advantage over the outsider.

6.4.5 Summary

Using a uniform password distribution drawn from a small password space (e.g. 4-digit PINs), the parameters of this protocol can be tuned to provide cho-

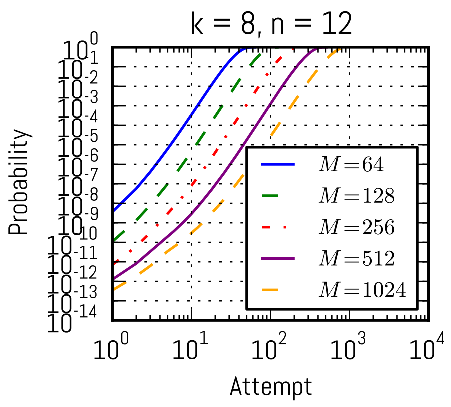


(a) Frequencies of sampled passwords.

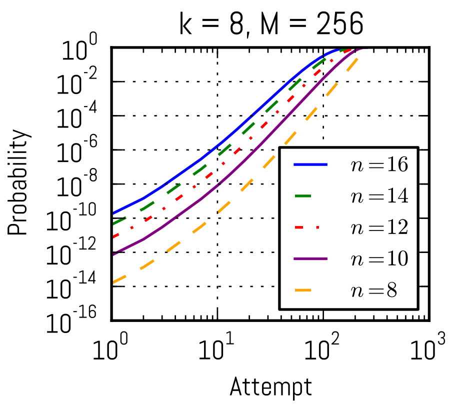


(b) PMFs of collision-rich hashes of sampled passwords.

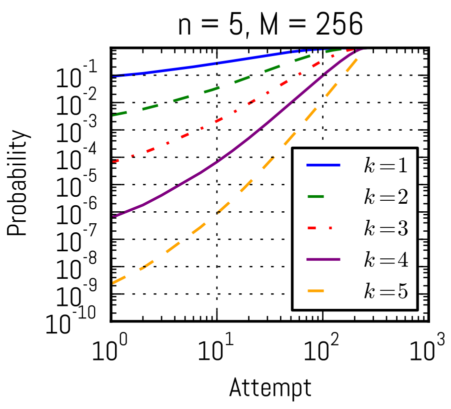
Figure 6.7: The RockYou distribution, sampled for several values of M.



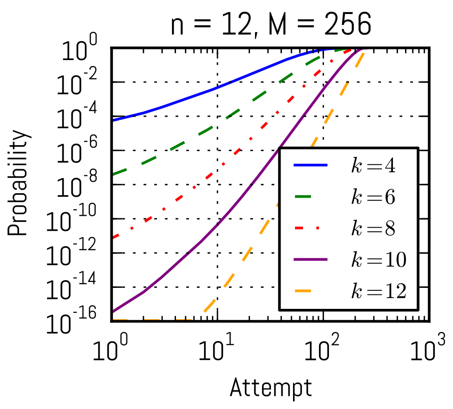
(a) Effect of varying M.



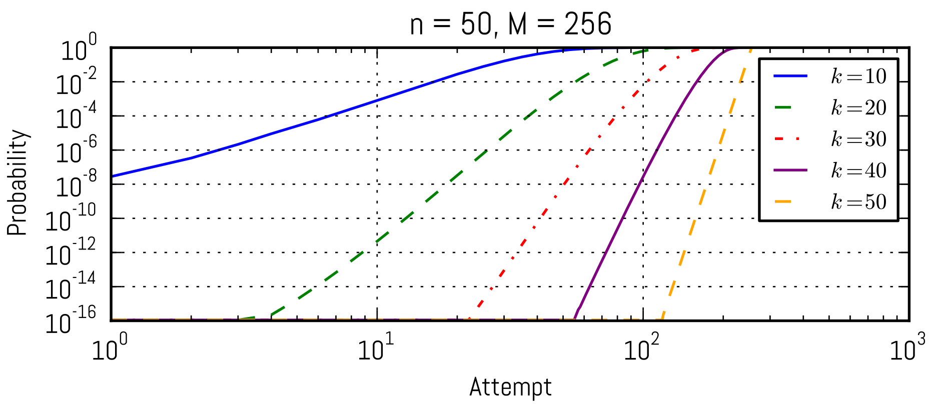
(b) Effect of varying n.



(c) Effect of varying k (small N).



(d) Effect of varying k (medium N).



(e) Effect of varying k (large N).

Figure 6.8: Attacking the sampled RockYou distribution.

sen resistance against malicious outsiders. Insiders have an advantage over outsiders, in that they have fewer potential passwords to check, but insider attack success probabilities on the order of 10^{-3} can be maintained over short time periods (approximately two days): even if a list of passwords to check is short, an online dictionary attack is very slow.

If user Bob chooses his own password, outsider Mallory is in a slightly better position, as he can attack password hashes in order of likelihood. The use of collision-rich hashes confounds him, however, flattening the probability distribution of the various possible h_M values so that an attack success probability of 10^{-4} can be maintained over extended periods of time. Interestingly, in this case the insider has almost no advantage over the outsider: a user-chosen password from a real distribution may be a better defense against insiders simply because it is drawn from a much larger space of possible passwords. Of course, this advantage may not hold if the distribution of user-chosen passwords is known to be weaker than that contained in the RockYou data set, but empirical evaluation of password selection within different communities is left to others.

6.5 REVOCATION

Any scheme that employs public-key cryptography must provide a mechanism for revoking keys. In the case of Footlights, revocation should be very infrequently-required, but the subject must be addressed nonetheless.

As described in Section 6.2, a Footlights user can create a constrained, time-limited key for purposes such as vacations, but in fact, the delgation of limited authority is a generally-wise practice that Footlights client software should perform transparently. Short-term, purpose-limited signing and encryption keys should only be accepted because they are signed by longer-term keys and so up, up a hierarchy to the user's "root" private key. This delegation of authority ensures that the damage that can be done by a leaked key is limited and that — with one important exception — there is always a higher-level authorisation key available to declare a lower-level key invalid.

The exceptional case is that of the user's "root" private key. Compromise of such keys should be very infrequent: they should only rarely be manipulated in the clear, on the user's most personal devices. If a user's root key is compromised, however, existing techniques such as the Online Certificate Status Protocol (OCSP) [262] can be used to query revocation state from any server — or servers — the user identifies. Revocation might be presented to users as a "permanently delete account" command, implemented as a signature of a revocation message with the private key in question.

6.6 RELATED WORK

Distributed protocols for authentication have existed since Needham and Schroeder described their seminal 1978 protocol [190]. Until this work, however, password authentication has required that the supplicant either trust the authentication server not to perform offline dictionary attacks or choose a password strong enough to resist such attacks.

The Needham–Schroeder protocol inspired the Kerberos authentication protocol as described by Neuman [192], which in its original form was vulnerable to offline dictionary attack by impersonators and network eavesdroppers. This weakness led to Kerberos "pre-authentication" in later versions of the protocol [263]. In the original protocol, a supplicant asked the authorisation server (AS) for a ticket-granting ticket (TGT, a cryptographic capability) without providing any authentication data. The server responded with a TGT encrypted under a key derived from the user's password. A user with knowledge of the password could decrypt this ciphertext to obtain the TGT, but so could an attacker performing an offline dictionary attack against the user's password. Kerberos pre-authentication allows the AS to challenge the supplicant, requiring them to submit evidence of password knowledge, e.g. the current time encrypted under the user's key. This prevents an impersonator from trivially obtaining the material required to perform a dictionary attack but an eavesdropper can still obtain the timestamp encrypted under the user's password and attack it instead. Furthermore, the AS must be trusted with the user's password-derived key.

In 1993, Gong et al. considered the problem of protecting "poorly chosen secrets" from dictionary attack when using network protocols [118]. These protocols, including a variant of the Kerberos protocol, were designed to protect user passwords from eavesdroppers on the network, but they were still based on the assumption that centralised servers would be trusted with the user's password. This is not compatible with the trust requirements of Footlights: no principal apart from the user should need to be trusted with the user's password.

Bellovin and Merritt's original Encrypted Key Exchange (EKE) protocol allowed two principals to negotiate a session key based on knowledge of a shared secret without exposing the negotiation to a dictionary attack by an eavesdropper [43]. More precisely, an eavesdropper would be forced to mount a dictionary attack against a randomly-generated key pair rather than the password. Instead of sending plaintext encrypted by the shared secret, the protocol is initiated by sending a public key encrypted with the shared secret. If the public key is encoded to be indistinguishable from random, an eavesdropper will not be able to perform an offline dictionary attack against the secret. This protected both the password itself and the session key from computationally feasible attacks by eavesdroppers, but it required both parties to store the password; an attacker capable of compromising one host or the other could learn the password and impersonate either party.

Augmented Encrypted Key Exchange (A-EKE), by the same authors, performed the same protocol with a one-way hash[3] of the password $h(p)$ substituted for the password p, followed by an additional proof of knowledge of p by one principal [44]. A-EKE is more explicitly a client–server protocol: a stolen $h(p)$ allows an attacker to impersonate one principal but not the other. This protocol provides strong authentication properties, but it is not suitable for the Footlights case: an attacker who steals $h(p)$ — or a malicious authentication agent who is given it — can perform an offline dictionary attack against it to learn p, so any party who possesses a verifier must be trusted.

Lucks' Open Key Encryption (OKE) protocol allows principals to negotiate a session key based on knowledge of a shared secret, as in EKE, but OKE does not require a new public-private key pair to be generated for every negotiation [166]. When Alice initiates the protocol, instead of sending an ephemeral

[3]This "password-hashing operation" need not be a cryptographic hash function: in a public-key–based model, p can be a private key and $h(p)$ the corresponding public key.

key encrypted under the shared secret, she sends a public key E and a nonce m. Bob replies with his own nonce μ and a proof of knowledge of the secret (a hash of E, m, μ and the shared secret) that has been combined with a strong randomly-chosen secret encrypted under E. This combination can be done with any invertible group operation: Alice will be able to retrieve the encrypted strong secret only if she can invert the operation using her knowledge of E, m, μ and the shared weak secret. This strong secret can be used to generate a session key. Like EKE, the OKE protocol trusts both principals to hold the shared secret (e.g. a user-chosen password), so it is not suitable for the Footlights setting.

Another closely-related protocol is Jablon's SPEKE (simple password exponential key exchange) [135]. This protocol is effectively a Diffie-Hellman key exchange [90], but instead of a public generator g as the base for the session key $k = g^{xy}$, a function f is parameterised by the secret password, so $k = f(p)^{xy}$. Like EKE, SPEKE has an extended variant in which only one party needs to know the password; this is called B-SPEKE [134]. Like A-EKE, this is a *verifier* protocol: the additional proof of password knowledge can be verified by the party that does not hold the password. B-SPEKE uses a Diffie-Hellman variant for its verification stage: the verifying party creates a challenge g^X based on a public generator g and a random value X. Knowledge of the password p allows the supplicant to reply with $\left(g^X\right)^p = g^{Xp}$. The verifying party holds a *verifier* g^p, allowing it to test that $\left(g^p\right)^X = g^{Xp}$. As in A-EKE, this prevents supplicant impersonation, but the password can still be attacked using a stolen verifier as a password oracle: for candidate password p', an attacker can test if $g^{p'} = g^p$.

Wu's secure remote password (SRP) protocol [236] is inappropriate for the Footlights scenario for the same reason. SRP provides one-way or mutual authentication using modular-exponentiation–based password verification. Like A-EKE and B-SPEKE, it provides no protection against offline dictionary attacks by insiders. SRP is a verifier-based protocol, and a verifier is a password oracle that can be queried offline. SRP discards the password-derived hash $x = h(\text{salt}, P)$ because Wu describes it as "equivalent to the plaintext password P", but for the purposes of an offline dictionary attack, so is the verifier $v = g^x$ that SRP requires. A hash and an exponentiation have different computational costs, but they are both one-way operations that serve as password oracles. SRP could treat this exponentiation differently from a salted hash if it were done with respect to a small modulus, creating a collision-rich operation as this chapter describes, but SRP prescribes exponentiation in the conventional manner: with respect to a large prime number. To introduce intentional collisions leads inexorably to a multi-party protocol such as the one described in this chapter: collisions make online attacks easier, so they must be balanced by a requirement to carry out several successful attacks.

The cryptographic community has contributed further work to this problem, later known as Password-Authenticated Key Exchange (PAKE): proving the correctness of existing protocols, as in the work of Bellare et al. [42], as well as proposing new theoretically-grounded generic constructions as in the work of Catalano et al. [69]. In all of these cases, however, a password verifier can be used to conduct an offline dictionary attack, just as in earlier work.

ISO 11770-4 also contains techniques for authenticating with weak secrets: "balanced" password-authenticated key agreement, augmented key agreement and password-authenticated key retrieval [25]. As above, however, nothing prevents a malicious verifier from conducting an offline dictionary attack against the weak secret.

Laurie's Nigori is a protocol for storing secrets such as Web passwords on remote servers [154]. It is conceptually an online password vault, where users authenticate to the vault with one password-derived key and store secrets that have been encrypted and MACed with other password-derived keys. It is assumed that the three keys (authentication, encryption and MAC) cannot be used to determine the user's master password; that is, that the password is resistant to offline dictionary attack. In contrast, the distributed authentication protocol presented in this chapter is designed to provide reasonable security properties even with weak passwords. Nigori allows a secret sharing scheme to be used with several severs in order to reduce the "trust or burden" required of any one of them. Nigori's description notes that "this is likely to be a poor defence unless [the user] uses different passwords at different servers". This weakness could be overcome, however, by an authentication protocol that uses collision-rich hashes, such as the one in this chapter.

Another related vein of cryptographic work is Stinson's *universal hash-*

ing [220], which randomises the use of hash functions in order to reduce the probability of collision among adversary-supplied keys. The work described in this chapter is a kind of inverse of universal hashing: rather than reducing collisions, my distributed authentication protocol defeats adversaries by intentionally introducing collisions that confound attacks.

Lomas and Christianson have previously used collision-rich hash functions with weak passwords for integrity checks on trusted computing bases [162]. In the Lomas–Christianson protocol, a user powers on a stateless workstation, which retrieves an OS kernel across an untrusted network and boots it. Before the kernel boots, its integrity is checked by hashing it together with the user's password using a collision-rich hash function and comparing the result to a user-specific public checksum stored alongside the kernel. Because the checksum incorporates a collision-rich hash of the user's password, the attacker cannot perform an offline dictionary attack to determine the password: he can only calculate a set of candidate passwords. This set of candidate passwords can be used to generate a set of checksums for the modified kernel, but the attacker has no way of knowing which one of these will be accepted. The use of collision-rich hash functions in this work has influenced the protocols in this chapter.

6.7 SUMMARY OF CONTRIBUTIONS

In this chapter, I have described a protocol that allows a user to authenticate to a cloud of peers using only a weak password (problem "Weak secrets" on page 143). This authentication can be done via any computer on which the user is willing to type his password (problem "Location independence" on page 143). Authentication agents, which are assumed to be honest-but-curious, are unable to learn the user's password via offline dictionary attack; they do not need to be trusted (problem "Limited trust" on page 144). Finally, attackers are unable to determine either which authentication agents provide their services to a particular user or which users are served by a particular authentication agent (problem “Plausible deniability” on page 144).

The weak password used in the protocol can be chosen from a uniformly random distribution over a small space (e.g. a 4-digit PIN) or by the user from a large password space. In the former case, resistance to outside attackers is excellent, providing low attack success probabilities over an extended period of time (months). It is significantly less resistant to insider attacks, however. In the latter case, a compromise is struck that reduces outsider attack resistance in exchange for the freedom and flexibility to choose one’s own password. In this case, insiders gain almost no advantage over outsiders when attacking the weak password. The sheer size of the password space means that even though an insider can narrow down the set of possible passwords by a factor of 64, 128 or even 1024, the remaining passwords must be checked as an outsider would: using an online dictionary attack.

This protocol allows users to retrieve a private key from untrusted agents with only weak passwords for authentication. The private key can be used to provide message integrity in the context of a decentralised social network such as Footlights, solving the larger *assertion problem*: the ability of users to assert who said what without relying on a trusted third party.

7

Conclusions

Today's online social networks (OSNs) provide users with an incredible array of affordances, allowing them to tap into a social realm wider than their physical circle of friends, facilitating the spread of information across continents and reducing the cost of group formation so much that bagpipe photography enthusiasts around the globe can discuss their passion more easily than members of a political party can assemble within a single constituency. The outcome of these networks is a more connected world, but that connectedness comes at a cost to confidentiality and user privacy.

7.1 Problems

Chapter 1 described the socio-technical problem that is privacy in online social networks, illustrating it with concrete scenarios. These scenarios featured user data in contemporary OSNs being shared more widely than the users involved wanted or expected it to be. I made the point that a technical system alone cannot "solve" privacy, but technical contributions could be made that would allow users to change their own privacy outcomes. Chapter 2 distilled several concrete technical problems out of the scenarios:

Intent (problem "Explicit expressions of user intent" on page 16) — sharing should be driven by users' expressions of intent.

Identity (problem "Multi-faceted identity" on page 17) — users should be able to present different facets of identity to different audiences without revealing the existence of other facets.

Availability (problem "High availability" on page 18) — the system must compete with commercial providers, so its availability (both uptime and bitrate) should be of commercial standard.

Trust (problem "Untrusted infrastructure" on page 18) — users should not need to trust any infrastructure to enforce their desired sharing policies.

Applications (problem "Social applications" on page 19) — the system should provide an API to allow extension of the social experience via rich applications that operate on user data indirectly by default.

Cost (problem "Cost" on page 20) — the cost of using the system should be exposed to users, who should have a choice of how to defray it without sacrificing privacy.

Anonymity (problem "Linkability and anonymity" on page 21) — users should be able to choose their own privacy–performance trade-offs: it should be possible to achieve some degree of anonymity at decreased performance.

I have demonstrated some of the ways today's OSNs do not meet these standards: they have exposed users' private information, sharing it more widely than users' expressed intent. Chapter 3 demonstrated practical threats to user privacy, in terms of both user data privacy and social graph privacy. The former class of threats is illustrated by a qualitative description of how user data has been leaked via changing default settings, advertiser access and application platforms. This was based on both literature survey and original research. The second class was illustrated with a quantitative analysis of how releasing a sampled subset of social graph information allows attackers to closely approximate important characteristics of the entire graph.

7.2 FOOTLIGHTS

In Chapter 4 I started to describe the Footlights system. Footlights is a privacy-enabling online social network that explores a new mix of centralised and distributed elements to provide users with performant functionality without paying a privacy penalty. Its name is derived from Goffman's description of social interaction as a theatrical performance [115] and is also a homage to the Cambridge Footlights theatrical troupe. Footlights is an architecture, a set of protocols and a proof-of-concept implementation that demonstrates the viability of a new kind of social network. The prototype code is available from `https://github.com/trombonehero/Footlights` under the Apache License [24].

Chapter 4 describes the sharable storage system that underpins Footlights. This system allows users to store private-by-default encrypted data as a sea of 4 kB ciphertext blocks on a commodity storage provider. Blocks are encrypted with block-specific symmetric keys; their content is only readable by principals which possess the relevant keys. Blocks in the encrypted store contain explicit links to other blocks, which are only visible to principals that possess the block's symmetric key. The choice of a content-addressed store (CAS) ensures that global caching will be efficient, never requiring freshness checks for static content, and that users will never need to download the same content twice or perform deep searches for fresh content on trees of already-seen data. The projected cost of operating the storage component of Footlights is less than one US dollar per user-year. The system also provides opportunities for covert communication, including a low-bitrate *perfectly unobservable* communications channel.

This global shared filesystem is exposed to social applications via an API described in Chapter 5, "Distributed Social Applications". Footlights is *generative* [248]: it provides an API that developers can use to craft as-yet-unimagined social applications. Users maintain control of their data: by default, applications operate indirectly on user data. Footlights applications are written on two platforms: a browser-based UI and a JVM-based backend. In the browser, UI elements are sandboxed, but manipulate a rich *visual context* using a proxy that resembles the JavaScript DOM API. This UI front-end can communicate with a corresponding back-end via an Ajax channel. The backend can be written in any JVM language. The Footlights API provides back-ends with access to the Footlights filesystem and local user files via the "security by designation" model described by Yee [241]: a photo-sharing application may read photos from the user's hard drive, but only because the user has selected which files to open in an "Open File" dialog. Encryption is handled at the level of the storage substrate, so even malicious applications are incapable of transparently leaking user data. With explicit user authorisation, applications can share content with each other or other users.

Finally, Chapter 6 describes a distributed authentication framework that allows users to access the Footlights distributed system from untrusted computers without losing control of their digital identity. Footlights ultimately relies on public-key cryptography for confidentiality and integrity, but problem "Multi-faceted identity" on page 17 requires that the system be accessible "without leveraging trusted authentication infrastructure or requiring users to carry hardware cryptographic tokens". To solve this problem, I describe a protocol that allows users to retrieve a strong secret (a private key) from a set of *authentication agents* using a weak secret (a random PIN or user-chosen password) without allowing the agents to determine the value of the secret, even with an offline dictionary attack.

Together, these three components of the Footlights system could be used to change the outcomes of all the scenarios given in Chapter 1.

In scenario "Jack" on page 3, a LiveJournal user named "Jack" had private details from his account re-posted as public ones, likely a result of password compromise. If "Jack" had used the Footlights distributed authentication protocol instead, his password would be safer from technical attacks. If, however, his problem was that a friend successfully guessed his password — a plausible explanation — the Footlights authentication protocol would not solve the problem. It would, however, provide him with a technical measure that could be imposed at a cost: "Jack" could forego the distributed authentication approach and carry his private key with him. This would allow him to use the storage system and application platform with high security, but the decision to adopt that approach is more economic than technical.

In scenario "CBSA" on page 4, photos and comments were posted to Facebook by new recruits of the Canada Border Services Agency and subsequently viewed by members of the public and the media. Facebook's default settings and privacy UI failed to capture the sharing intent of the CBSA recruits; Footlights would have allowed them to share the content as widely as they desired, but its private-by-default model would have made accidental disclosure of this kind less likely.

In scenario "Facebook" on page 4, Facebook imposed a policy change in which certain parts of a user's profile such as name, friends and connections became "public information". This change was undertaken despite the fact that some users had taken care to make this information private. In Footlights, such a policy change could never be enforced: sharing decisions are made by users, not the providers of centralised infrastructure.

In scenario "Application developers" on page 5, several developers of third-party Facebook applications were caught sharing private user data with data brokers, who in turn sold the information to their customers. In the Footlights application ecosystem, this sharing would not have been permitted by default: Footlights applications can only access user data with explicit user authorisation. Even if applications had asked for — and received — such authorisation, Footlights applications are *observable* : users would be able to detect the data leak from locally-excuted Footlights more easily than the Wall Street Journal did with remotely-executed Facebook applications.

Finally, scenario "Sophie" on page 5 describes the plight of an OSN non-user. "Sophie" is a skeptic: her concerns about the privacy practices of today's OSNs lead her to forego the benefits of social applications. "Sophie" could be a privacy advocate concerned about her personal information or a corporate consumer concerned about confidential competitor information. In either case, using Footlights instead of Facebook would allow "Sophie" to be assured that her confidentiality requirements are being met: rather than trusting a centralised provider to enforce her security policy, she could verify the cryptographic protocols and software implementation herself, or rely on an external auditor or certifier to do this on her behalf.

7.3 FUTURE WORK

The research described in this dissertation has revealed many paths that beg exploration, but I could not walk them all as one traveller. These include:

Implementation In this dissertation, I presented a design and prototype implementation. I argue that the system will scale well, but the proof of the pudding is in the eating. It is beyond the scope of this dissertation

to implement all of the APIs required to drive mass adoption, but it is my hope that publicly-available, royalty-free specifications and an open-source prototype will encourage adoption by vendors seeking to securely integrate online social elements into their platforms.
Specific implementation tasks include the integration of mutual authentication protocols as described in Section 4.1, the development of the inter-process communication primitives described in §5.3.4.3 and the implementation of the distributed authentication protocol in Chapter 6.

Security proofs Footlights relies on techniques whose security has not been proven in the standard model. Some difficulties can be obviated under the random oracle assumption, but proofs in the standard model would be a natural direction to pursue. It is desirable to prove that, first, convergent encryption provides indistinguishability when random padding is used and that, second, the construction in Section 4.8, "Covert communication" creates ciphertext blocks that are indistinguishable to an attacker who does not know the block's random padding or the covert content to be hidden within it.

User experience Footlights has been designed with security usability considerations in mind, but I am not a user experience designer. Footlights' aesthetic appeal also leaves much to be desired. User adoption of Footlights will require a focus on usability and aesthetics: if the system is not pleasing to use, it will not be used.

Funding Networks and storage substrates are not free. Current social networks are indirectly paid for by user data; future distributed systems will require alternative payment schemes. I have argued that the storage backend of Footlights could be supported by direct payment (e.g. via occasional premium text messages) or privacy-preserving targeted advertising. One important question is, how can such an advertising scheme be run without privileged access to the TCB? Today's advertisements rely on privilege in order to resist "click-fraud": they employ schemes such as frame-busting, which breaks out of browser containers to prevent encapsulation and redirection, in the same way that malware attempts to escape virtual machine encapsulation. If Footlights does not allow such techniques, how much will it deminish the perceived value of a targeted advertisement?

Consent How can researchers study social networks when users are in charge of their own data? Today, researchers have such ready access to large corpora that discussions of informed consent often do not happen. If users are given full control over their personal information, how can researchers obtain data and understand biases? These issues have been considered within the realm of traditional social sciences, but how can those lessons be applied to studies of millions of users, rather than dozens?

DTN routing Footlights creates a new kind of Delay-Tolerant Network (DTN), one that relies on one-way communication *and* opportunistic contact. This network also has more stringent privacy requirements than other DTNs: neither sender nor recipient should need to make their address globally known. What constitutes a good routing algorithm for such a DTN? Are there essential trade-offs between routing efficiency and information revelation by participants? Can classes of routing algorithms be tuned for the privacy of sender, recipient or intermediaries according to need?

DTN congestion The global utility of a covert DTN is derived from routing decisions made by individual users, each with their own goals and incentives. This problem sounds familiar, but it is not the well-known Tragedy of the Commons: in Footlights, each node has exclusive control over its own patch of the network. Rather, the problem resembles that of congestion on the Internet, but the solutions employed there (point-to-point peering agreements, entrusting network stacks to a few privileged individuals) will not apply to a fully-fledged distributed system like Footlights. How can costs and incentives be employed to make the Footlights covert DTN usable? Since no one principal will be in a position to set costs, how can the protocol be made self-enforcing?

Ecosystem The Footlights architecture and prototype constitute an extensible core for a new online social network. That network will only flourish if others use it and build on it. To succeed, Footlights must have an ecosystem of both users and application developers: people must be able to find their friends and have something to do together. This requires a more rigorous treatment of API expressibility and usability.

7.4 SUMMARY

"You're not the customer. The ad service buyer is the customer. You're the commodity."

`liorean`, forum user, on the 2004 introduction of GMail [158]

The claim is often made that, to use online social networks, users must give up their privacy. In this dissertation, I have argued that this cynicism is unwarranted: it is possible to build a new kind of online social network that lets users enjoy all the activities and functions they have grown accustomed to in today's OSNs, with comparable levels of performance, but without having to place absolute faith in the companies operating the disks and networks.

Privacy and performance need not be forever locked in conflict. It is possible to *rely on* centralised infrastructure to provide the availability that users expect without *trusting* it to enforce users' intentions for sharing. I have described an architecture and prototype implementation of a hybrid centralised–distributed OSN called Footlights that allows users to choose their own privacy and performance trade-offs *and* enjoy the benefits of social applications. Source code is available from `https://github.com/trombonehero/Footlights` under the open-source Apache License, version 2.0 [24].

The cost of operating Footlights' centralised storage foundation is expected to be less than one US dollar per user-year, recoupable directly or via privacy-preserving advertising.

This alternative business model is not guaranteed to succeed: it is hard to get anyone to join an empty social network. What Footlights demonstrates, however, is that this alternative model is possible. The way that today's OSNs do business is just one way of doing business. Privacy is not inherently incompatible with social networking.

A

CONTENT DELIVERY NETWORKS

The argument for a system like Footlights is predicated on the plausibility of the economics: Footlights can only work if someone pays for it. The plausibility of the Footlights cost structure is addressed in problem "Cost" on page 20 in Chapter 4, "Sharable storage". This appendix contains raw data concerning the costs of the commodity providers that the Footlights storage system could be built on.

In this appendix, I show the advertised costs of storage and data transfer on several providers' platforms. Three providers (Amazon, Google and Rackspace) provide combined storage and content distribution platforms. A fourth provider, NetDNA, specialises in content delivery services; if using a pure CDN like NetDNA, content must be backed by storage from another party.

The costs of Amazon's Simple Storage Service (S3) and CloudFront delivery service [328] are shown in Table A.1. Amazon is unique among the surveyed providers in that it charges its clients according to storage volume, data transfer volume *and* number of HTTP PUT and GET requests which act on stored data. Since Footlights divides files into small (4 kB) blocks, this HTTP request cost is the most significant portion of the potential cost of running the Footlights storage system on Amazon (see problem "Cost" on page 20).

Table A.2 shows the cost of storage and content delivery using the Google App Engine platform [336]. This cost structure is very simple: there is a small quota of free usage, above which a fixed rate applies. This structure is simple, but the marginal rate for both storage and data transfer is high.

Rackspace [330] has an even simpler cost structure than Google: it eliminates the quota of no-charge usage. As shown in Table A.3, there is one marginal rate

Table A.1: Pricing of Amazon S3 and CloudFront services.

(a) Storage

Volume	Marginal cost
Up to 1 TB	\$125 / TB·month
1–50 TB	\$105 / TB·month
50–500 TB	\$95 / TB·month
500 TB–1 PB	\$90 / TB·month
1–5 PB	\$80 / TB·month
Above 5 PB	\$55 / TB·month

(b) HTTP requests

Request type	Cost
GET	\$10 / Mrequest
PUT	\$1 / Mrequest

(c) Data transfer (out)

Volume per month	Marginal cost
Up to 1 GB	0
1 GB–10 TB	\$120 / TB
10–50 TB	\$90 / TB
50–150 TB	\$70 / TB
150–500 TB	\$50 / TB
Above 500 TB	not specified

Table A.2: Pricing of Google App Engine storage and content delivery.

(a) Storage

Volume	Marginal cost
Up to 5 GB	No charge
Above 5 GB	\$130 / TB·month

(b) Data transfer (out)

Volume per day	Marginal cost
Up to 1 GB	No charge
Above 1 GB	\$120 / TB

Table A.3: Pricing of Rackspace storage and content delivery.

(a) Storage

Volume	Marginal cost
Any volume	\$100 / TB·month

(b) Data transfer (out)

Volume per month	Marginal cost
Any volume	\$180 / TB

Table A.4: Pricing of NetDNA content delivery.

(a) Storage

Volume	Marginal cost
Not available	–

(b) Data transfer (out)

Volume per month	Marginal cost
Up to 10 GB	\$60 / TB
10–50 TB	\$50 / TB
50–150 TB	\$40 / TB
150–350 TB	\$35 / TB
350–650 TB	\$30 / TB
650–3 PB	\$20 / TB
Above 3 PB	\$10 / TB

for any volume of storage or data transfer.

Finally, Table A.4 shows the cost of delivering content via the NetDNA content delivery network [337]. These costs are much lower than comparable data transfer costs for any of the storage providers, but they do not include the cost of extracting the data from disks. Thus, to use a pure CDN in conjunction with a storage provider would result in some "double-billing": the storage provider would charge to provide data to the CDN and the CDN would charge to replicate and distribute it around the world.

Bibliography

[24] Apache license, version 2.0. *The Apache Software Foundation*, Jan. 2004. URL: `http://www.apache.org/licenses/LICENSE-2.0.html`.

[25] ISO 11770-4: Information technology — Security techniques — Key management — Part 4: Mechanisms based on weak secrets, June 2012. URL: `http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=39723`.

[26] A. Acquisti and R. Gross. Imagined communities: awareness, information sharing, and privacy on the Facebook. In D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, G. Danezis, and P. Golle, editors, *PET 2006: Proceedings of the 6th Workshop on Privacy Enhancing Technology*, pages 36–58. Springer, 2006. `doi:10.1007/11957454_3`.

[27] L. M. Aiello and G. Ruffo. LotusNet: tunable privacy for distributed online social network services. *Computer Communications*, 35(2012):75–88, Dec. 2010. `doi:10.1016/j.comcom.2010.12.006`.

[28] L. M. Aiello and G. Ruffo. Secure and flexible framework for decentralized social network services. In *PERCOMW 2010: Proceedings of the 8th IEEE International Conference on Pervasive Computing and Communications Workshops*, pages 594–599. IEEE, 2010. `doi:10.1109/PERCOMW.2010.5470506`.

[29] R. J. Anderson. Liability and computer security: nine principles. In *ESORICS '94: Proceedings of the Third European Symposium on Research in Computer Security*, pages 231–245, 1994. URL: `http://www.cl.cam.ac.uk/~rja14/Papers/liability.pdf`.

[30] R. J. ANDERSON. Why information security is hard — an economic perspective. In *ACSAC 2001: Proceedings of the 17th Annual Computer Security Applications Conference*, pages 358–365, 2001. doi:10.1109/ACSAC.2001.991552.

[31] R. J. ANDERSON, R. NEEDHAM, AND A. SHAMIR. The steganographic file system. In *IH '98: Proceedings of the Second International Hiding Workshop*, pages 73–82, 1998. doi:10.1007/3-540-49380-8_6.

[32] R. J. ANDERSON AND F. PETITCOLAS. On the limits of steganography. *IEEE Journal on Selected Areas in Communications*, 16(4):474–481, 1998. doi:10.1109/49.668971.

[33] P. K. ATREY. A secret sharing based privacy enforcement mechanism for untrusted social networking operators. In *MiFor '11: Proceedings of the 3rd International ACM Workshop on Multimedia in Forensics and Intelligence*, page 13. ACM Press, 2011. doi:10.1145/2072521.2072525.

[34] A. BACK, U. MÖLLER, AND A. STIGLIC. Traffic analysis attacks and trade-offs in anonymity providing systems. In *IH 2001: Proceedings of the 4th International Workshop on Information Hiding*, pages 245–257. Springer, 2001. doi:10.1007/3-540-45496-9_18.

[35] L. BACKSTROM, C. DWORK, AND J. KLEINBERG. Wherefore art thou r3579x?: anonymized social networks, hidden patterns and structural steganography. *WWW '07: Proceedings of the 16th International Conference on the World Wide Web*, pages 181–190, 2007. URL: http://www2007.org/paper232.php.

[36] R. BADEN, A. BENDER, N. SPRING, B. BHATTACHARJEE, AND D. STARIN. Persona: an online social network with user-defined privacy. In *SIGCOMM 2009: Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*. ACM, Aug. 2009. doi:10.1145/1592568.1592585.

[37] F. BEATO, M. KOHLWEISS, AND K. WOUTERS. Scramble! your social network data. In *PETS 2011: Proceedings of the 11th International Sym-*

posium on Privacy Enhancing Technology, 2011. URL: http://www.cosic.esat.kuleuven.be/publications/article-2029.pdf.

[38] A. Beimel and B. Chor. Secret sharing with public reconstruction (extended abstract). In *CRYPTO '95: Proceedings of the 15th Annual International Conference on Advances in Cryptology*, page 366, 1995. URL: http://www.springerlink.com/content/agrjh8ug4vdd7chk/.

[39] M. Bellare, A. Boldyreva, and A. O'Neill. Deterministic and efficiently searchable encryption. In *CRYPTO 2007: Proceedings of the 27th Annual International Cryptology Conference on Advances in Cryptology*. Springer-Verlag, Aug. 2007. URL: http://portal.acm.org/citation.cfm?id=1777777.1777820.

[40] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption. In *Foundations of Computer Science, 1997. Proceedings., 38th Annual Symposium on*, pages 394–403, 1997. doi:10.1109/SFCS.1997.646128.

[41] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among notions of security for public-key encryption schemes. *Advances in Cryptology—CRYPTO'98*, pages 26–45, 1998. doi:10.1007/BFb0055718.

[42] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques*, pages 139–155, 2000. URL: http://www.iacr.org/archive/eurocrypt2000/1807/18070140-new.pdf.

[43] S. Bellovin and M. Merritt. Encrypted key exchange: password-based protocols secure against dictionary attacks. In *SP 1992: Proceedings of the 13th IEEE Symposium on Security and Privacy*, pages 72–84, 1992. doi:10.1109/RISP.1992.213269.

[44] S. M. Bellovin and M. Merritt. Augmented encrypted key exchange: a password-based protocol secure against dictionary attacks and pass-

word file compromise. In *CCS '93: Proceedings of the 1st ACM Conference on Computer and Communications security*. ACM, Dec. 1993. doi:10.1145/168588.168618.

[45] A. BENJAMIN. Predicting and postdicting the effects of word frequency on memory. *Memory & Cognition*, 31(2):297–305, 2003. URL: http://www.springerlink.com/index/PR6407K858N57462.pdf.

[46] A. BESMER, H. R. LIPFORD, M. SHEHAB, AND G. CHEEK. Social applications: exploring a more secure framework. In *SOUPS '09: Proceedings of the Fifth Symposium on Usable Privacy and Security*, 2009. doi:10.1145/1572532.1572535.

[47] K. BIBA. Integrity considerations for secure computer systems. Technical Report MTR-3153, MITRE Corporation, June 1975. URL: http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA039324.

[48] M. BILENKO, M. RICHARDSON, AND J. Y. TSAI. Targeted, not tracked: client-side solutions for privacy-friendly behavioral advertising. In *HotPETs 2011: Hot Topics in Privacy Enhancing Technologies*, pages 1–13, May 2011. URL: http://petsymposium.org/2011/papers/hotpets11-final3Bilenko.pdf.

[49] D. BINDEL, Y. CHEN, S. CZERWINSKI, P. EATON, D. GEELS, R. GUMMADI, S. RHEA, H. WEATHERSPOON, C. WELLS, B. ZHAO, J. KUBIATOWICZ, D. BINDEL, Y. CHEN, S. CZERWINSKI, P. EATON, D. GEELS, R. GUMMADI, S. RHEA, H. WEATHERSPOON, C. WELLS, B. ZHAO, J. KUBIATOWICZ, D. BINDEL, Y. CHEN, S. CZERWINSKI, P. EATON, D. GEELS, R. GUMMADI, S. RHEA, H. WEATHERSPOON, C. WELLS, AND B. ZHAO. OceanStore: an architecture for global-scale persistent storage. *ACM SIGOPS Operating Systems Review*, 34(5):190–201, Dec. 2000. doi:10.1145/384264.379239.

[50] A. BLACKWELL, C. BRITTON, A. COX, T. GREEN, C. GURR, G. KADODA, M. KUTAR, M. LOOMES, C. NEHANIV, AND M. PETRE. Cognitive dimen-

sions of notations: design tools for cognitive technology. *Cognitive Technology 2001*, LNAI(2117):325–341, 2001. doi:10.1007/3-540-44617-6_31.

[51] M. Blaze. A cryptographic file system for UNIX. In *CCS '93: Proceedings of the 1st ACM Conference on Computer and Communications security*. ACM, Dec. 1993. doi:10.1145/168588.168590.

[52] O. Bodriagov and S. Buchegger. Encryption for peer-to-peer social networks. In *PASSAT 2011: Proceedings of the Third IEEE International Conference on Privacy, Security, Risk and Trust*, pages 1302–1309, 2011. doi:10.1109/PASSAT/SocialCom.2011.158.

[53] D. Boneh, C. Gentry, and B. Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In *CRYPTO 2005: Proceedings of the 25th Annual International Conference on Advances in Cryptology*, Aug. 2005. URL: http://www.springerlink.com/index/374pjlcakffdffnw.pdf.

[54] J. Bonneau. The science of guessing: analyzing an anonymized corpus of 70 million passwords. In *SP 2012: Proceedings of the 34th IEEE Symposium on Security and Privacy*, May 2012. doi:10.1109/SP.2012.49.

[55] J. Bonwick, M. Ahrens, V. Henson, M. Maybee, and M. Shellenbaum. The Zettabyte file system. In *FAST 2003: 2nd Usenix Conference on File and Storage Technologies (Work-in-Progress session)*, 2003. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.184.3704.

[56] N. Borisov, I. Goldberg, and E. Brewer. Off-the-record communication, or, why not to use PGP. In *WPES '04: Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society*. ACM, Oct. 2004. doi:10.1145/1029179.1029200.

[57] N. Borisov, I. Goldberg, and D. Wagner. Intercepting mobile communications: the insecurity of 802.11. In *MobiCom '01: Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*. ACM, July 2001. doi:10.1145/381677.381695.

[58] D. M. BOYD. *Taken out of context — American teen sociality in networked publics*. PhD thesis, UC Berkeley, 2008. URL: http://www.danah.org/papers/TakenOutOfContext.pdf.

[59] D. M. BOYD AND E. HARGITTAI. Facebook privacy settings: who cares? *First Monday*, 15(8), Aug. 2010. URL: http://firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/view/3086/2589.

[60] V. BOYKO. On the security properties of OAEP as an all-or-nothing transform. IN M. WIENER, editor, *CRYPTO '99: Proceedings of the 19th Annual International Conference on Advances in Cryptology*, pages 503–518. Springer, Dec. 1999. doi:10.1007/3-540-48405-1_32.

[61] S. BRAGHIN, V. IOVINO, G. PERSIANO, AND A. TROMBETTA. Secure and policy-private resource sharing in an online social network. In *IEEE International Conference on Social Computing*, Oct. 2011. URL: http://libeccio.dia.unisa.it/Papers/SecOSN/.

[62] U. BRANDES. A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, 25(2):163–177, 2001. doi:10.1080/0022250X.2001.9990249.

[63] E. BREWER. Lessons from giant-scale services. *IEEE Internet Computing*, 5(4):46–55, 2001. doi:10.1109/4236.939450.

[64] J. BROWN, V. J. LEWIS, AND A. F. MONK. Memorability, word frequency and negative recognition. *Quarterly Journal of Experimental Psychology*, 29(3):461–473, Aug. 1977. doi:10.1080/14640747708400622.

[65] S. BUCHEGGER, D. SCHIOBERG, L.-H. VU, AND A. DATTA. PeerSoN: P2P social networking: early experiences and insights. In *SNS '09: Proceedings of the Second ACM EuroSys Workshop on Social Network Systems*. ACM, Mar. 2009. doi:10.1145/1578002.1578010.

[66] J. BUSCA, F. PICCONI, AND P. SENS. Pastis: a highly-scalable multi-user peer-to-peer file system. In *Euro-Par 2005: Proceedings of the 11th International Euro-Par Conference on Parallel Processing*, volume LNCS 3648, pages

1173–1182. Springer, 2005. URL: `http://www.springerlink.com/index/dfye6ftnfpy51qac.pdf`.

[67] P. R. Carpentier, J. F. Van Riel, and T. Teugels. Content addressable information encapsulation, representation, and transfer. *United States Patent and Trademark Office*, 713/165; 173/170; 713/176; 713/180; 713/193; 380/28; 705/51(09/236,366), Oct. 2004. URL: `http://www.google.com/patents/US6807632`.

[68] M. Castro and B. Liskov. Practical Byzantine fault tolerance. In *OSDI '99: Proceedings of the 3rd USENIX Symposium on Operating Systems Design and Implementation*, pages 173–186. ACM, 1998. URL: `http://www.usenix.org/events/osdi99/full_papers/castro/castro_html/node5.html`.

[69] D. Catalano, D. Pointcheval, and T. Pornin. IPAKE: isomorphisms for password-based authenticated key exchange. In D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, and M. Franklin, editors, *CRYPTO 2004: Proceedings of the 24th Annual International Conference on Advances in Cryptology*, pages 477–493. Springer Berlin Heidelberg, 2004. `doi:10.1007/978-3-540-28628-8{\_}29`.

[70] D. L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2), Feb. 1981. `doi:10.1145/358549.358563`.

[71] D. L. Chaum, A. Fiat, and M. Naor. Untraceable electronic cash. In *CRYPTO '88: Proceedings of the Eighth Annual International Conference on Advances in Cryptology*, pages 319–327. Springer, 1990. URL: `http://www.springerlink.com/index/vfjy8u0f7byftqx9.pdf`.

[72] H.-Y. Chien and J.-K. Jan. New hierarchical assignment without public key cryptography. *Computers and Security*, 22(6):523–526, Sept. 2003. `doi:10.1016/S0167-4048(03)00613-8`.

[73] B. Christianson. Definition of trust. Personal Communication, July 2012.

[74] B. Christianson and W. S. Harbison. Why isn't trust transitive? In *SPW '96: Proceedings of the Fourth International Workshop on Security Protocols*, pages 171–176, 1996. doi:10.1007/3-540-62494-5_16.

[75] J. Claessens, C. Diaz, R. Faustinelli, and B. Preneel. A secure and privacy-preserving web banner system for targeted advertising. COmputer Security and Industrial Cryptography (COSIC), KU Leuven, 2003. URL: http://www.cosic.esat.kuleuven.be/publications/article-824.pdf.

[76] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: a distributed anonymous information storage and retrieval system. In H. Federrath, editor, *Designing Privacy Enhancing Technologies — Proceedings of the International Workshop on Design Issues in Anonymity and Unobservability*, volume LNCS 2009, pages 46–66. Springer, 2000. doi:10.1007/3-540-44702-4_4.

[77] B. Cohen. Incentives build robustness in BitTorrent. In *Workshop on Economics of Peer-to-Peer Systems*, May 2003. URL: http://www.ittc.ku.edu/~niehaus/classes/750-s06/documents/BT-description.pdf.

[78] C. Conrado, M. Petkovic, M. van der Veen, and W. van der Velde. Controlled sharing of personal content using digital rights management. *Journal of Research and Practice in Information Technology*, 38(1):85–96, 2006. URL: http://ws.acs.org.au/jrpit/JRPITVolumes/JRPIT38/JRPIT38.1.85.pdf.

[79] L. Cutillo, R. Molva, and T. Strufe. Safebook: a privacy-preserving online social network leveraging on real-life trust. *IEEE Communications Magazine*, 47(12):94–101, 2009. doi:10.1109/MCOM.2009.5350374.

[80] L. A. Cutillo, R. Molva, and T. Strufe. Privacy preserving social networking through decentralization. In *WONS 2009: Proceedings of the*

Sixth Interational Conference on Wireless On-Demand Network Systems and Services, pages 145–152, 2009. `doi:10.1109/WONS.2009.4801860`.

[81] L. A. Cutillo, R. Molva, and T. Strufe. Safebook: feasibility of transitive cooperation for privacy on a decentralized social network. In *AOC 2009: Third International IEEE WoWMoM Workshop on Autonomic and Opportunistic Communications*, pages 1–6, 2009. `doi:10.1109/WOWMOM.2009.5282446`.

[82] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb. Social coding in GitHub: transparency and collaboration in an open software repository. In *CSCW '12: Proceedings of the 2012 ACM Conference on Computer Supported Cooperative Work*. ACM, Feb. 2012. `doi:10.1145/2145204.2145396`.

[83] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *SOSP '01: Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*. ACM, Dec. 2001. `doi:10.1145/502034.502054`.

[84] G. Danezis, R. Dingledine, and N. Mathewson. Mixminion: design of a type III anonymous remailer protocol. In *SP 2003: Proceedings of the 24th IEEE Symposium on Privacy and Security*, pages 2–15, 2003. `doi:10.1109/SECPRI.2003.1199323`.

[85] A. P. de Barros. O futuro dos backdoors. In *CNASI 2005: Congresso de Auditoria de Sistemas, Segurança da Informação e Governança*. CISSP-ISSAP, Sept. 2005. URL: `http://www.paesdebarros.com.br/backdoors.pdf`.

[86] M. Dell'Amico, P. Michiardi, and Y. Roudier. Password strength: an empirical analysis. In *INFOCOM 2010: Proceedings of the 29th IEEE Conference on Computer Communications*, pages 1–9, 2010. `doi:10.1109/INFCOM.2010.5461951`.

[87] D. E. Denning. An intrusion-detection model. *IEEE Transactions on Software Engineering*, SE-13(2):222–232, 1987. `doi:10.1109/TSE.1987.232894`.

[88] D. E. Denning and P. J. D. M. D. Schwartz. The tracker: a threat to statistical database security. *ACM Transactions on Database Systems*, 4(1), Mar. 1979. doi:10.1145/320064.320069.

[89] C. Diaz, S. Seys, J. Claessens, and B. Preneel. Towards measuring anonymity. In *PET 2002: Second International Workshop on Privacy Enhancing Technologies*, pages 184–188, 2002. URL: http://www.springerlink.com/content/3qb837jkpgukc6b5/.

[90] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976. doi:10.1109/TIT.1976.1055638.

[91] R. Dingledine and N. Mathewson. Tor: the second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, 2004. URL: http://portal.acm.org/citation.cfm?id=1251396.

[92] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, Mar. 1983. doi:10.1109/TIT.1983.1056650.

[93] J. R. Douceur. The Sybil attack. In *International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002. URL: http://www.springerlink.com/index/3an0ek5gfan3dtx9.pdf.

[94] J. R. Douceur, A. Adya, W. J. Bolosky, D. Simon, and M. Theimer. Reclaiming space from duplicate files in a serverless distributed file system. Technical Report MSR-TR-2002-30, July 2002. URL: http://research.microsoft.com/apps/pubs/default.aspx?id=69954.

[95] J. R. Douceur, A. Adya, W. J. Bolosky, P. Simon, and M. Theimer. Reclaiming space from duplicate files in a serverless distributed file system. In *ICDCS 2002: Proceedings of the 22nd International Conference on Distributed Computing Systems*, pages 617–624, 2002. doi:10.1109/ICDCS.2002.1022312.

[96] C. Dwork. Differential privacy. In *ICALP 2006: 33rd International Colloquium on Automata, Languages and Programming*, volume LNCS 4052, 2006. URL: `http://www.springerlink.com/index/383p21xk13841688.pdf`.

[97] R. W. Emerson. Worship. In *The Conduct of Life*, page 192. Boston, Ticknor and Fields, 1860. URL: `http://archive.org/details/conductlife00emerrich`.

[98] J. Epstein, J. McHugh, R. Pascale, H. Orman, G. Benson, C. Martin, A. Marmor-Squires, B. Danner, and M. Branstad. A prototype B3 trusted X Window System. In *ACSAC 1991: Proceedings of the Seventh Annual Computer Security Applications Conference*, pages 44–55, 1991. `doi:10.1109/CSAC.1991.213019`.

[99] K. Fall. A delay-tolerant network architecture for challenged internets. In *SIGCOMM '03: Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. ACM, Aug. 2003. `doi:10.1145/863955.863960`.

[100] K. Fall and S. Farrell. DTN: an architectural retrospective. *IEEE Journal on Selected Areas in Communications*, 26(5):828–836, 2008. URL: `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4530739`.

[101] L. Fang and K. LeFevre. Privacy wizards for social networking sites. In *WWW '10: Proceedings of the 19th International World Wide Web Conference*. ACM, Apr. 2010. `doi:10.1145/1772690.1772727`.

[102] R. Feiertag and P. G. Neumann. The foundations of a provably secure operating system (PSOS). In *NCC '79: Proceedings of the 1979 AFIPS National Computer Conference*, 1979. `doi:10.1109/AFIPS.1979.116`.

[103] A. Felt and D. Evans. Privacy protection for social networking platforms. In *W2SP 2008: Web 2.0 Security and Privacy 2008*, May 2008. URL: `http://w2spconf.com/2008/`.

[104] A. Fiat and M. Naor. Broadcast encryption. In *CRYPTO '93: Proceedings of the 13th Annual International Conference on Advances in Cryp-*

tology, pages 480–491, 1993. URL: `http://www.springerlink.com/index/C4CDHFU88JLLW2PX.pdf`.

[105] M. H. FISCHER, T. J. PURTELL, AND M. S. LAM. Email clients as decentralized social apps in Mr. Privacy. In *HotPETs 2011: Hot Topics in Privacy Enhancing Technologies*, pages 1–10, May 2011. URL: `http://petsymposium.org/2011/papers/hotpets11-final1Fischer.pdf`.

[106] B. FORD, P. SRISURESH, AND D. KEGEL. Peer-to-peer communication across network address translators. In *Proceedings of the 2005 USENIX Annual Technical Conference*. USENIX Association, 2005. URL: `http://dl.acm.org/citation.cfm?id=1247360.1247373`.

[107] L. R. FORD AND D. R. FULKERSON. Flows in networks. Technical Report R-375-PR, RAND Corporation, Aug. 1962. URL: `http://www.rand.org/pubs/reports/R375.html`.

[108] L. FREEMAN. A set of measures of centrality based on betweenness. *Sociometry*, 40(1):35–41, Mar. 1977. URL: `http://www.jstor.org/stable/10.2307/3033543`.

[109] K. E. FU. *Group sharing and random access in cryptographic storage file systems*. PhD thesis, Massechusetts Institute of Technology, May 1999. URL: `http://www.cs.umass.edu/~kevinfu/papers/fu-masters.pdf`.

[110] D. K. GIFFORD, P. JOUVELOT, M. A. SHELDON, J. W. O'TOOLE, AND JR. Semantic file systems. In *SOSP '91: Proceedings of the Thirteenth ACM Symposium on Operating Systems Principles*. ACM, Oct. 1991. `doi:10.1145/121132.121138`.

[111] D. K. GIFFORD, R. M. NEEDHAM, AND M. D. SCHROEDER. The Cedar file system. *Communications of the ACM*, 31(3):288–298, Mar. 1988. `doi:10.1145/42392.42398`.

[112] H. GIFFORD. A delectable dreame. In *A posie of gilloflowers eche differing from other in colour and odour, yet all sweete*. Thomas Dawson,

1580. URL: http://quod.lib.umich.edu/e/eebo/A01740.0001.001/1:38?rgn=div1;view=fulltext.

[113] S. GILBERT AND N. LYNCH. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2), June 2002. doi:10.1145/564585.564601.

[114] M. GLADWELL. *The tipping point: how little things can make a big difference*. Back Bay Books, Jan. 2002. URL: http://www.amazon.com/dp/0316346624.

[115] E. GOFFMAN. *The presentation of self in everyday life*. Anchor Books, Jan. 1959. URL: http://www.worldcat.org/title/presentation-of-self-in-everyday-life/oclc/35136526.

[116] O. GOLDREICH, S. MICALI, AND A. WIGDERSON. How to play ANY mental game. In *STOC '87: Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, pages 218–229. ACM, Jan. 1987. doi:10.1145/28395.28420.

[117] L. GONG. Java security architecture revisited. *Communications of the ACM*, 54(11), Nov. 2011. doi:10.1145/2018396.2018411.

[118] L. GONG, M. A. LOMAS, R. M. NEEDHAM, AND J. H. SALTZER. Protecting poorly chosen secrets from guessing attacks. *IEEE Journal on Selected Areas in Communications*, 11(5):648–656, 1993. doi:10.1109/49.223865.

[119] L. GONG, M. MUELLER, H. PRAFULLCHANDRA, AND R. SCHEMERS. Going beyond the sandbox: an overview of the new security architecture in the Java Development Kit 1.2. *USITS '97: Proceedings of the USENIX Symposium on Internet Technologies and Systems*, 1997. URL: http://static.usenix.org/publications/library/proceedings/usits97/full_papers/gong/gong.pdf.

[120] B. GOPAL AND E. AL. Integrating content-based access mechanisms with hierarchical file systems. In *OSDI '99: Proceedings of the 3rd USENIX Symposium on Operating Systems Design and Implementation*,

1999. URL: `https://www.usenix.org/conference/osdi-99/integrating-content-based-access-mechanisms-hierarchical-file-systems`.

[121] K. Graffi, P. Mukherjee, B. Menges, D. Hartung, A. Kovacevic, and R. Steinmetz. Practical security in P2P-based social networks. In *LCN 2009: Proceedings of the 34th Annual IEEE Conference on Local Computer Networks*, pages 269–272, 2009. `doi:10.1109/LCN.2009.5355085`.

[122] T. Green and M. Petre. Usability analysis of visual programming environments: a 'cognitive dimensions' framework. *Journal of visual languages and computing*, 7(2):131–174, 1996. URL: `http://www.sciencedirect.com/science/article/pii/S1045926X96900099`.

[123] B. Greschbach, G. Kreitz, and S. Buchegger. The Devil is in the metadata — new privacy challenges in decentralised online social networks. In *SESOC 2012: Proceedings of the 4th IEEE International Workshop on Security and and Social Networking*, Mar. 2012. URL: `http://www.peerson.net/papers/sesocMetaPrivacy.pdf`.

[124] D. Grolimund, L. Meisser, S. Schmid, and R. Wattenhofer. Cryptree: a folder tree structure for cryptographic file systems. *SRDS '06: Proceedings of the 25th IEEE Symposium on Reliable Distributed Systems*, pages 189–198, 2006. `doi:10.1109/SRDS.2006.15`.

[125] R. Gross and A. Acquisti. Information revelation and privacy in online social networks. In *WPES '05: Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society*. ACM, Nov. 2005. `doi:10.1145/1102199.1102214`.

[126] S. Guha, B. Cheng, and P. Francis. Privad: practical privacy in online advertising. In *NSDI '11: Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation*. USENIX Association, Mar. 2011. URL: `http://portal.acm.org/citation.cfm?id=1972457.1972475`.

[127] S. Guha, K. Tang, and P. Francis. NOYB: privacy in online social networks. In *WOSN '08: Proceedings of the First Workshop on Online Social Networks*. ACM, Aug. 2008. `doi:10.1145/1397735.1397747`.

[128] P. GÜHRING. Concepts against man-in-the-browser attacks. Technical report, Jan. 2007. URL: `http://www.cacert.at/svn/sourcerer/CAcert/SecureClient.pdf`.

[129] R. GUTTENTAG AND D. CARROLL. Memorability judgments for high- and low-frequency words. *Memory & Cognition*, 26(5):951–958, Sept. 1998. `doi:10.3758/BF03201175`.

[130] N. HARDY. The confused deputy (or why capabilities might have been invented). *ACM SIGOPS Operating Systems Review*, 22(4):36–38, Oct. 1988. URL: `http://dl.acm.org/citation.cfm?id=54289.871709`.

[131] C. HERLEY. The plight of the targeted attacker in a world of scale. In *WEIS '10: The Ninth Workshop on the Economics of Information Security*, May 2010.

[132] J.-H. HOEPMAN. The ephemeral pairing problem. IN T. KANADE, J. KITTLER, J. M. KLEINBERG, F. MATTERN, J. C. MITCHELL, M. NAOR, O. NIERSTRASZ, C. PANDU RANGAN, B. STEFFEN, M. SUDAN, D. TERZOPOULOS, D. TYGAR, M. Y. VARDI, G. WEIKUM, AND A. JUELS, editors, *FC '04: Proceedings of the Eigth International Conference on Financial Cryptography*, pages 212–226. Springer, 2004. `doi:10.1007/978-3-540-27809-2_22`.

[133] G. HORNUNG AND C. SCHNABEL. Data protection in Germany I: the population census decision and the right to informational self-determination. *Computer Law & Security Review*, 25(1):84–88, Jan. 2009. `doi:10.1016/j.clsr.2008.11.002`.

[134] D. JABLON. Extended password key exchange protocols immune to dictionary attack. In *WETICE-1997: Proceedings of the Sixth IEEE Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 248–255, 1997. `doi:10.1109/ENABL.1997.630822`.

[135] D. P. JABLON. Strong password-only authenticated key exchange. *ACM SIGCOMM Computer Communication Review*, 26(5):5–26, Oct. 1996. `doi:10.1145/242896.242897`.

[136] T. N. Jagatic, N. A. Johnson, M. Jakobsson, and F. Menczer. Social phishing. *Communications of the ACM*, 50(10), Oct. 2007. doi:10.1145/1290958.1290968.

[137] A. Juels. Targeted advertising... and privacy too. *Topics in Cryptology—CT-RSA 2001*, LNCS 2020:408–424, 2001. URL: http://www.springerlink.com/index/G5WB298DWW81DKR0.pdf.

[138] M. Just and D. Aspinall. Challenging challenge questions. In *Trust 2009*, Feb. 2009. URL: http://homepages.inf.ed.ac.uk/mjust/Trust2009.pdf.

[139] M. Just and D. Aspinall. Personal choice and challenge questions: a security and usability assessment. In *SOUPS '09: Proceedings of the Fifth Symposium on Usable Privacy and Security*, 2009.

[140] J. Katz and Y. Lindell. *Introduction to modern cryptography*. Principles and Protocols. Chapman & Hall/CRC, Aug. 2007. URL: http://books.google.co.uk/books?id=TTtVKHdOcDoC.

[141] T. J. Killian. Processes as files. In *USENIX Summer Conference*, 1984.

[142] H. Kim, J. Tang, and R. J. Anderson. Social authentication: harder than it looks. In *Financial Cryptography*, Feb. 2012. URL: http://www.cl.cam.ac.uk/~rja14/Papers/socialauthentication.pdf.

[143] J. J. Kistler and M. Satyanarayanan. Disconnected operation in the Coda file system. In *SOSP '91: Proceedings of the Thirteenth ACM Symposium on Operating Systems Principles*. ACM, Oct. 1991. doi:10.1145/121132.121166.

[144] P. Kocher, J. Jaffe, B. Jun, C. Laren, and N. Lawson. Self-protecting digital content. Technical report, Cryptography Research International, 2002. URL: http://www.cryptography.com/public/pdf/SelfProtectingContent.pdf.

[145] A. Korolova. Privacy violations using microtargeted ads: a case study. In *ICDMW 2010: Proceedings of the 10th IEEE International Conference on Data Mining Workshops*, pages 474–482, 2010. doi:10.1109/ICDMW.2010.137.

[146] A. Korolova. Privacy violations using microtargeted ads: a case study. *Journal of Privacy and Confidentiality*, 3(1), 2011. URL: http://repository.cmu.edu/jpc/vol3/iss1/3/.

[147] B. Krishnamurthy and C. E. Wills. Characterizing privacy in online social networks. In *WOSN '08: Proceedings of the First Workshop on Online Social Networks*, Aug. 2008. doi:10.1145/1397735.1397744.

[148] B. Krishnamurthy and C. E. Wills. On the leakage of personally identifiable information via online social networks. In *WOSN '09: Proceedings of the Second ACM Workshop on Online Social Networks*. ACM, Aug. 2009. doi:10.1145/1592665.1592668.

[149] P. Kumar and M. Satyanarayanan. Supporting application-specific resolution in an optimistically replicated file system. In *WWOS 1993: Proceedings of the Fourth Workshop on Workstation Operating Systems*, pages 66–70, 1993. doi:10.1109/WWOS.1993.348170.

[150] P. Kumar and M. Satyanarayanan. Flexible and safe resolution of file conflicts. In *Proceedings of the 1995 USENIX Annual Technical Conference*, 1995. URL: http://www.cs.cmu.edu/~coda/docdir/usenix95.pdf.

[151] L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982. doi:10.1145/357172.357176.

[152] B. W. Lampson. Dynamic protection structures. In *AFIPS '69 (Fall): Proceedings of the AFIPS 1969 Fall Joint Computer Conference*. ACM, Nov. 1969. doi:10.1145/1478559.1478563.

[153] B. W. Lampson. A note on the confinement problem. *Communications of the ACM*, 16(10), Oct. 1973. doi:10.1145/362375.362389.

[154] B. Laurie. Nigori: storing secrets in the cloud [online]. May 2010. URL: http://www.links.org/files/nigori-overview.pdf.

[155] J. Li, M. Krohn, D. Mazieres, and D. Shasha. Secure untrusted data repository (SUNDR). In *OSDI '04: Proceedings of the 6th USENIX Symposium on Operating Systems Design and Implementation*, 2004. URL: http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=pubmed&cmd=Retrieve&dopt=AbstractPlus&list_uids=7548375546647870391related:t_swEZE3wWgJ.

[156] M. Li, B. Alessio, and W. Zhou. OST: a transaction based online social trust model for social network and file sharing security. In *EUC 2010: Proceedings of the 8th IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*, pages 826–832. IEEE, 2010. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5703616.

[157] J. Lindamood, R. Heatherly, M. Kantarcioglu, and B. Thuraisingham. Inferring private information using social network data. In *WWW '09: Proceedings of the 18th International World Wide Web Conference*. ACM, Apr. 2009. doi:10.1145/1526709.1526899.

[158] liorean. Google 1G mail [online]. Feb. 2004. URL: http://www.codingforums.com/showpost.php?p=206569&postcount=9.

[159] S. B. Lipner. A comment on the confinement problem. In *SOSP '75: Proceedings of the Fifth ACM Symposium on Operating Systems Principles*. ACM, Nov. 1975. doi:10.1145/800213.806537.

[160] S. B. Lipner, W. Wulf, R. Schell, G. Popek, P. G. Neumann, C. Weissman, and T. Linden. Security kernels. *AFIPS '74: Proceedings of the 1974 National Computer Conference and Exposition*, May 1974. URL: http://dl.acm.org/citation.cfm?id=1500361.

[161] J. Loeliger. *Version control with Git*. O'Reilly, June 2009. URL: http://www.worldcat.org/title/version-control-with-git/oclc/297148669.

[162] M. Lomas and B. Christianson. Remote booting in a hostile world: to whom am I speaking? *IEEE Computer*, 28(1):50–54, 1995. doi:10.1109/2.362630.

[163] P. A. Loscocco and S. D. Smalley. Integrating flexible support for security policies into the Linux operating system. In *FREENIX 2001: Proceedings of the FREENIX Track of the 2001 USENIX Annual Technical Conference*, pages 29–42, 2001.

[164] R. Love. Get on the D-BUS. *Linux Journal*, 2005(130):3, 2005. URL: http://www.linuxjournal.com/article/7744.

[165] M. M. Lucas and N. Borisov. FlyByNight: mitigating the privacy risks of social networking. In *WPES '08: Proceedings of the 7th ACM Workshop on Privacy in the Electronic Society*. ACM, Oct. 2008. doi:10.1145/1456403.1456405.

[166] S. Lucks. Open key exchange: how to defeat dictionary attacks without encrypting public keys. In *SPW '98: Proceedings of the Sixth International Workshop on Security Protocols*, 1998. doi:10.1007/BFb0028161.

[167] W. Luo, Q. Xie, and U. Hengartner. FaceCloak: an architecture for user privacy on social networking sites. In *CSE '09: Proceedings of the 12th IEEE International Conference on Computational Science and Engineering*, pages 26–33, 2009. doi:10.1109/CSE.2009.387.

[168] M. Mannan and P. C. van Oorschot. Privacy-enhanced sharing of personal content on the web. In *WWW '08: Proceedings of the 17th International World Wide Web Conference*. ACM, Apr. 2008. doi:10.1145/1367497.1367564.

[169] E. Maximilien, T. Grandison, K. Liu, T. Sun, D. Richardson, and S. Guo. Enabling privacy as a fundamental construct for social networks. In *CSE '09: Proceedings of the 12th IEEE International Conference on Computational Science and Engineering*, pages 1015–1020, 2009. doi:10.1109/CSE.2009.431.

[170] D. Mazieres, M. Kaminsky, M. F. Kaashoek, E. Witchel, D. Mazieres, M. Kaminsky, M. F. Kaashoek, and E. Witchel. Separating key management from file system security. In *SOSP '99: Proceedings of the Seventeenth ACM Symposium on Operating Systems Principles*. ACM, 1999. doi:10.1145/319151.319160.

[171] D. Mazieres, M. Kaminsky, M. F. Kaashoek, E. Witchel, D. Mazieres, M. Kaminsky, M. F. Kaashoek, and E. Witchel. Separating key management from file system security. *ACM SIGOPS Operating Systems Review*, 33(5):124–139, Dec. 1999. doi:10.1145/319344.319160.

[172] A. McDonald and M. G. Kuhn. StegFS: a steganographic file system for Linux. In A. Pfitzmann, editor, *IH '99: Proceedings of the Third International Workshop on Information Hiding*, pages 463–477, 2000.

[173] M. K. McKusick, W. N. Joy, S. J. Leffler, and R. S. Fabry. A fast file system for UNIX. *ACM Transactions on Computer Systems (TOCS)*, 2(3), Aug. 1984. doi:10.1145/989.990.

[174] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Oct. 1996. URL: http://cacr.uwaterloo.ca/hac/.

[175] R. C. Merkle. A certified digital signature. In G. Brassard, editor, *Advances in Crypology — Proceedings of CRYPTO '89*, pages 218–238. Springer, 1990. doi:10.1007/0-387-34805-0{_}21.

[176] A. Mettler, D. Wagner, and T. Close. Joe-E: a security-oriented subset of Java. In *NDSS '10: Proceedings of the Network and Distributed System Security Symposium*, 2010.

[177] M. S. Miller. Securing ECMAScript 5. In *goto;*, page 71, Oct. 2010. URL: http://es-lab.googlecode.com/files/securing-es5.pdf.

[178] M. S. Miller. ECMAScript 5, Caja and retrofitting security. InfoQ, Feb. 2011. URL: http://www.infoq.com/interviews/ecmascript-5-caja-retrofitting-security.

[179] Y. MINSKY. OCaml for the masses. *Communications of the ACM*, 54(11), Nov. 2011. doi:10.1145/2018396.2018413.

[180] A. MISLOVE, A. POST, P. DRUSCHEL, AND K. P. GUMMADI. Ostra: leveraging trust to thwart unwanted communication. In *NSDI '08: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, pages 15–30, 2008. URL: http://static.usenix.org/event/nsdi08/tech/mislove.html.

[181] A. MISLOVE, B. VISWANATH, K. P. GUMMADI, AND P. DRUSCHEL. You are who you know: inferring user profiles in online social networks. In *WSDM '10: Proceedings of the Third ACM International Conference on Web Search and Data Mining*. ACM, Feb. 2010. doi:10.1145/1718487.1718519.

[182] S. J. MURDOCH AND G. DANEZIS. Low-cost traffic analysis of Tor. In *SP 2005: Proceedings of the 26th IEEE Symposium on Security and Privacy*, pages 183–195, 2005. doi:10.1109/SP.2005.12.

[183] A. MUTHITACHAROEN, R. MORRIS, T. GIL, AND B. CHEN. Ivy: a read/write peer-to-peer file system. In *OSDI '02: Proceedings of the 5th USENIX Symposium on Operating Systems Design and Implementation*, Dec. 2002. URL: http://www.usenix.org/events/osdi02/tech/full_papers/muthitacharoen/muthitacharoen_html/.

[184] A. MUTHITACHAROEN, R. MORRIS, T. M. GIL, AND B. CHEN. Ivy: a read/write peer-to-peer file system. *ACM SIGOPS Operating Systems Review*, 36(SI):31, Dec. 2002. doi:10.1145/844128.844132.

[185] A. C. MYERS AND B. LISKOV. A decentralized model for information flow control. In *SOSP '97: Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*. ACM, Dec. 1997. URL: http://www.pmg.lcs.mit.edu/papers/iflow-sosp97.pdf.

[186] S. NAGARAJA. The economics of covert community detection and hiding. In *WEIS '09: The Eighth Workshop on the Economics of Information Security*, 2008. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.145.6548&rep=rep1&type=pdf.

[187] A. Narayanan, S. Barocas, V. Toubiana, H. Nissenbaum, and D. Boneh. A critical look at decentralized personal data architectures. In *DUMW: Data Usage Management on the Web*, Apr. 2012. URL: `http://dig.csail.mit.edu/2012/WWW-DUMW/papers/dumw2012_submission_5.pdf`.

[188] A. Narayanan and V. Shmatikov. How to break anonymity of the Netflix prize dataset. *arXiv*, Nov. 2007. URL: `http://arxiv.org/pdf/cs/0610105`.

[189] A. Narayanan and V. Shmatikov. Robust de-anonymization of large sparse datasets. In *SP 2008: Proceedings of the 29th IEEE Symposium on Security and Privacy*, pages 111–125, May 2008. `doi:10.1109/SP.2008.33`.

[190] R. M. Needham and M. D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12), Dec. 1978. `doi:10.1145/359657.359659`.

[191] R. M. Needham, R. D. Walker, R. M. Needham, and R. D. Walker. The Cambridge CAP computer and its protection system. *ACM SIGOPS Operating Systems Review*, 11(5):1–10, Nov. 1977. `doi:10.1145/800214.806541`.

[192] B. Neuman and T. Ts'o. Kerberos: an authentication service for computer networks. *IEEE Communications Magazine*, 32(9):33–38, 1994. URL: `http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=312841`.

[193] M. Odersky, P. Altherr, V. Cremet, B. Emir, S. Maneth, S. Micheloud, N. Mihaylov, M. Schinz, E. Stenman, and M. Zenger. An overview of the Scala programming language. Technical Report IC/2004/64, 2004. URL: `http://infoscience.epfl.ch/record/52656`.

[194] R. H. Patterson, G. A. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka. Informed prefetching and caching. In *SOSP '95: Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*. ACM, Dec. 1995. `doi:10.1145/224056.224064`.

[195] B. PAWLOWSKI, C. JUSZCZAK, P. STAUBACH, C. SMITH, D. LEBEL, AND D. HITZ. NFS version 3 design and implementation. In *USENIX Summer Conference*, pages 1–16, 1994.

[196] P. PEARCE, A. FELT, AND G. NUNEZ. AdDroid: privilege separation for applications and advertisers in Android. In *ASIACCS '12: Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, 2012. URL: `http://www.cs.berkeley.edu/~daw/papers/addroid-asiaccs12.pdf`.

[197] T. PERRIN. Public key distribution through "cryptoIDs". In *NSPW '03: Proceedings of the 2003 Workshop on New Security Paradigms*, pages 87–102, Aug. 2003. URL: `http://portal.acm.org/citation.cfm?id=986655.986669`.

[198] T. PEVNY, T. FILLER, AND P. BAS. Using high-dimensional image models to perform highly undetectable steganography. In *IH 2010: Proceedings of the 13th International Workshop on Information Hiding*, Jan. 2010. URL: `http://boss.gipsa-lab.grenoble-inp.fr/Warming/Materials/HUGO-electronic_pre_proc.pdf`.

[199] R. PIKE, D. PRESOTTO, K. THOMPSON, H. TRICKEY, AND P. WINTERBOTTOM. The use of name spaces in Plan 9. In *EW 5: Proceedings of the 5th ACM SIGOPS European Workshop: Models and Paradigms for Distributed Systems Structuring*. ACM, Sept. 1992. `doi:10.1145/506378.506413`.

[200] R. PIKE, D. PRESOTTO, K. THOMPSON, H. TRICKEY, AND P. WINTERBOTTOM. The use of name spaces in Plan 9. *ACM SIGOPS Operating Systems Review*, 27(2):72–76, Apr. 1993. `doi:10.1145/155848.155861`.

[201] C. G. PLAXTON, R. RAJARAMAN, AND A. W. RICHA. Accessing nearby copies of replicated objects in a distributed environment. In *SPAA '97: Proceedings of the Ninth Annual ACM Symposium on Parallel Algorithms and Architectures*. ACM, June 1997. `doi:10.1145/258492.258523`.

[202] B. Preneel. *Analysis and design of cryptographic hash functions*. PhD thesis, Katholieke Universiteit te Leuven, Feb. 1993. URL: `http://homes.esat.kuleuven.be/~preneel/phd_preneel_feb1993.pdf`.

[203] S. Quinlan and S. Dorward. Venti: a new approach to archival storage. In *Proceedings of the FAST 2002 Conference on File and Storage Technologies*, Jan. 2002. URL: `http://db.usenix.org/events/fast02/quinlan.html`.

[204] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. ACM, Aug. 2001. `doi:10.1145/383059.383072`.

[205] J. Raymond. Traffic analysis: protocols, attacks, design issues, and open problems. In *Designing Privacy Enhancing Technologies — Proceedings of the International Workshop on Design Issues in Anonymity and Unobservability*, pages 10–29, 2000.

[206] M. K. Reiter and A. D. Rubin. Crowds: anonymity for web transactions. *ACM Transactions on Information and System Security (TISSEC)*, 1(1), Nov. 1998. `doi:10.1145/290163.290168`.

[207] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz. Pond: the OceanStore prototype. In *FAST '03: Proceedings of the 2nd USENIX Conference on File and Storage Technologies*, pages 1–14, 2003. URL: `http://www.usenix.org/events/fast03/tech/rhea/rhea_html/`.

[208] R. L. Rivest. All-or-nothing encryption and the package transform. In E. Biham, editor, *FSE '97: Proceedings of the Fourth International Workshop on Fast Software Encryption*, pages 210–218. Springer, 1997. `doi:10.1007/BFb0052348`.

[209] R. L. RIVEST, A. SHAMIR, AND L. ADLEMAN. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2), Feb. 1978. doi:10.1145/359340.359342.

[210] J. H. SALTZER, D. P. REED, AND D. CLARK. End-to-end arguments in system design. *ACM Transactions on Computer Systems (TOCS)*, 2(4), Nov. 1984. doi:10.1145/357401.357402.

[211] S. SCELLATO, C. MASCOLO, M. MUSOLESI, AND J. CROWCROFT. Track globally, deliver locally: improving content delivery networks by tracking geographic social cascades. In *WWW 2011: Proceedings of the 20th International World Wide Web Conference*, Mar. 2011. URL: http://www.cl.cam.ac.uk/research/srg/netos/papers/www2011_geocascades.pdf.

[212] S. SCHECHTER, S. EGELMAN, AND R. REEDER. It's not what you know, but who you know: a social approach to last-resort authentication. *CHI '09: Proceedings of the 27th international conference on Human factors in computing systems*, Apr. 2009. URL: http://portal.acm.org/citation.cfm?id=1518701.1519003.

[213] S. SCHECHTER AND R. W. REEDER. 1 + 1 = You: measuring the comprehensibility of metaphors for configuring backup authentication. In *SOUPS '09: Proceedings of the Fifth Symposium on Usable Privacy and Security*, 2009.

[214] M. D. SCHROEDER AND J. H. SALTZER. A hardware architecture for implementing protection rings. *Communications of the ACM*, 15(3), Mar. 1972. doi:10.1145/361268.361275.

[215] W. SHAKESPEARE. *As you like it*. 1600. Act II, Scene 7.

[216] A. SHAMIR. How to share a secret. *Communications of the ACM*, 22(11), Nov. 1979. doi:10.1145/359168.359176.

[217] R. E. SMITH. Constructing a high assurance mail guard. In *NCSC 1994: Proceedings of the 17th National Computer Security Conference*, pages 247–

253. Secure Computer Corporation, 1994. URL: http://www.cryptosmith.com/docs/mailguard.pdf.

[218] M. Stiegler, A. H. Karp, K.-P. Yee, T. Close, and M. S. Miller. Polaris: virus-safe computing for Windows XP. *Communications of the ACM*, 49(9), Sept. 2006. doi:10.1145/1151030.1151033.

[219] M. Stiegler and M. Miller. A capability based client: the DarpaBrowser. *Combex Inc.*, BAA-00-06-SNK:1–101, June 2002.

[220] D. Stinson. Universal hashing and authentication codes. In J. Feigenbaum, editor, *Advances in Cryptology — Proceedings of CRYPTO '91*, pages 74–85. Springer, 1992. doi:10.1007/3-540-46766-1{_}5.

[221] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. ACM, Aug. 2001. doi:10.1145/383059.383071.

[222] N. Tolia, M. Kozuch, M. Satyanarayanan, B. Karp, T. Bressoud, and A. Perrig. Opportunistic use of content addressable storage for distributed file systems. In *Proceedings of the 2003 USENIX Annual Technical Conference*, pages 127–140, 2003. URL: http://www.usenix.org/events/usenix2003/tech/full_papers/tolia/tolia_html/.

[223] A. Tootoonchian, K. K. Gollu, S. Saroiu, Y. Ganjali, and A. Wolman. Lockr: social access control for web 2.0. In *WOSN '08: Proceedings of the First Workshop on Online Social Networks*. ACM, Aug. 2008. doi:10.1145/1397735.1397746.

[224] V. Toubiana, A. Narayanan, D. Boneh, H. Nissenbaum, and S. Barocas. Adnostic: privacy preserving targeted advertising. In *NDSS '10: Proceedings of the Network and Distributed System Security Symposium*, 2010. URL: http://www.isoc.org/isoc/conferences/ndss/10/pdf/05.pdf.

[225] C. Ungureanu, B. Atkin, A. Aranya, S. Gokhale, S. Rago, G. Całkowski, C. Dubnicki, and A. Bohra. HydraFS: a high-throughput file system for the HYDRAstor content-addressable storage system. In *FAST 2010: Proceedings of the 2010 USENIX Conference on File and Storage Technologies*. USENIX Association, Feb. 2010. URL: http://static.usenix.org/event/fast10/tech/.

[226] C. Vance and R. N. M. Watson. Security enhanced BSD. Technical report, Network Associates Laboratories, July 2003. URL: http://www.trustedbsd.org/sebsd-july2003.pdf.

[227] D. Wagner and D. Tribble. A security analysis of the Combex DarpaBrowser architecture. Technical report, 2002. URL: http://combexin.temp.veriohosting.com/papers/darpa-review/security-review.pdf.

[228] S. D. Warren and L. D. Brandeis. The right to privacy. *Harvard Law Review*, 4(5):193–220, Dec. 1890. URL: http://www.jstor.org/stable/1321160.

[229] M. Weir, S. Aggarwal, M. Collins, and H. Stern. Testing metrics for password creation policies by attacking large sets of revealed passwords. In *CCS '10: Proceedings of the 17th ACM Conference on Computer and Communications Security*. ACM, Oct. 2010. doi:10.1145/1866307.1866327.

[230] A. Whitten. *Making security usable*. PhD thesis, Carnegie Mellon University, 2004. URL: http://gaudior.net/alma/MakingSecurityUsable.pdf.

[231] A. Whitten and J. D. Tygar. Why Johnny can't encrypt: a usability evaluation of PGP 5.0. In *Proceedings of the 8th USENIX Security Symposium*, 1999. URL: http://usenix.org/events/sec99/whitten.html.

[232] Z. Wilcox-O'Hearn and B. Warner. Tahoe: the least-authority filesystem. In *StorageSS '08: Proceedings of the 4th ACM International Workshop on Storage Security and Survivability*. ACM, Oct. 2008. doi:10.1145/1456469.1456474.

[233] G. Wondracek, T. Holz, E. Kirda, and C. Kruegel. A practical attack to de-anonymize social network users. In *SP 2010: Proceedings of the 31st IEEE Symposium on Security and Privacy*, pages 223–238, 2010. doi: 10.1109/SP.2010.21.

[234] F.-L. Wong and F. Stajano. Multi-channel protocols. In B. Christianson, B. Crispo, J. A. Malcolm, and M. Roe, editors, *SPW '07: Proceedings of the Fifteenth International Workshop on Security Protocols*, pages 112–127. Springer, 2007. doi:10.1007/978-3-540-77156-2.

[235] A. Wool. A note on the fragility of the "Michael" message integrity code. *IEEE Transactions on Wireless Communications*, 3(5):1459–1462, 2004. doi: 10.1109/TWC.2004.833470.

[236] T. Wu. The secure remote password protocol. In *NDSS '98: Proceedings of the Network and Distributed System Security Symposium*, 1998. URL: http://www.isoc.org/isoc/conferences/ndss/98/wu.pdf.

[237] W. Xu, X. Zhou, and L. Li. Inferring privacy information via social relations. In *DEBSM 2008: Data Engineering for Blogs, Social Media, and Web 2.0*, pages 525–530, 2008. doi:10.1109/ICDEW.2008.4498373.

[238] J. Yan, A. Blackwell, R. J. Anderson, and A. Grant. The memorability and security of passwords – some empirical results. Technical Report UCAM-CL-TR-500, University of Cambridge, Sept. 2000. URL: http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-500.pdf.

[239] J. Yan, A. Blackwell, R. J. Anderson, and A. Grant. Password memorability and security: empirical results. *IEEE Security and Privacy Magazine*, 2(5):25–31, 2004. doi:10.1109/MSP.2004.81.

[240] S. Yardi, N. Feamster, and A. Bruckman. Photo-based authentication using social networks. In *WOSN '08: Proceedings of the First Workshop on Online Social Networks*. ACM, Aug. 2008. doi:10.1145/1397735.1397748.

[241] K.-P. Yee. Aligning security and usability. *IEEE Security and Privacy Magazine*, 2(5):48–55, 2004. doi:10.1109/MSP.2004.64.

[242] C. Yong, W. Jiangjiang, M. Songzhu, W. Zhiying, J. Ma, R. Jiangchun, and Y. Ke. Mailbook: privacy-protecting social networking via email. In *ICIMCS '11: Proceedings of the Third International Conference on Internet Multimedia Computing and Service*. ACM, Aug. 2011. doi:10.1145/2043674.2043700.

[243] H. Yu, P. Gibbons, and M. Kaminsky. SybilLimit: a near-optimal social network defense against Sybil attacks. In *SP 2008: Proceedings of the 29th IEEE Symposium on Security and Privacy*, 2008. doi:10.1109/SP.2008.13.

[244] H. Yu, M. Kaminsky, and P. Gibbons. SybilGuard: defending against Sybil attacks via social networks. In *SIGCOMM 2006: Proceedings of the ACM SIGCOMM 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 2006. URL: http://portal.acm.org/citation.cfm?id=1159913.1159945.

[245] J. Zhan. Secure collaborative social networks. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 40(6):682–689, 2010. doi:10.1109/TSMCC.2010.2050879.

[246] E. Zheleva and L. Getoor. To join or not to join: the illusion of privacy in social networks with mixed public and private user profiles. In *WWW '09: Proceedings of the 18th International World Wide Web Conference*. ACM, Apr. 2009. Poster. doi:10.1145/1526709.1526781.

[247] Y. Zhu, Z. Hu, H. Wang, H. Hu, and G.-J. Ahn. A collaborative framework for privacy protection in online social networks. In *CollaborateCom 2010: Proceedings of the 6th International Conference on Collaborative Computing: Networking, Applications and Worksharing*, pages 1–10, 2010. URL: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5766999.

[248] J. L. ZITTRAIN. *The future of the Internet (and how to stop it)*. Yale University Press, 2008. URL: `http://www.worldcat.org/isbn/978-0-300-15124-4`.

Other References

Standards and Specifications

[249] Trusted computer system evaluation criteria. Department of Defense, Dec. 1985. 5200.28-STD. URL: `http://csrc.nist.gov/publications/history/dod85.pdf`.

[250] New I/O APIs for the Java platform, May 2002. URL: `http://www.jcp.org/en/jsr/detail?id=51`.

[251] ECMAScript language specification. ECMA, June 2011. ECMA-262. URL: `http://www.ecma-international.org/publications/standards/Ecma-262.htm`.

[252] Facebook Query Language (FQL) [online]. Jan. 2012. URL: `https://developers.facebook.com/docs/reference/fql/`.

[253] Graph API [online]. June 2012. URL: `https://developers.facebook.com/docs/reference/api/`.

[254] V. Apparao, S. Byrne, M. Champion, S. Isaacs, A. Le Hors, G. Nicol, J. Robie, P. Sharpe, B. Smith, J. Sorensen, R. Sutor, R. Whitmer, and C. Wilson. Document Object Model (DOM) level 1 specification. W3C: World Wide Web Consortium, Oct. 1998. REC-DOM-Level-1-19981001. URL: `http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/`.

[255] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifier (URI): generic syntax. RFC 3986 (Standard), Jan. 2005. URL: `http://www.ietf.org/rfc/rfc3986.txt`.

[256] T. Berners-Lee, L. Masinter, and M. McCahill. Uniform Resource Locators (URL). RFC 1738 (Proposed Standard), Dec. 1994. Obsoleted

by RFCs 4248, 4266, updated by RFCs 1808, 2368, 2396, 3986, 6196, 6270. URL: http://www.ietf.org/rfc/rfc1738.txt.

[257] N. BORENSTEIN AND N. FREED. MIME (Multipurpose Internet Mail Extensions): mechanisms for specifying and describing the format of Internet message bodies. RFC 1341 (Proposed Standard), June 1992. Obsoleted by RFC 1521. URL: http://www.ietf.org/rfc/rfc1341.txt.

[258] D. CROCKFORD. The application/json media type for JavaScript Object Notation (JSON). RFC 4627 (Informational), July 2006. URL: http://www.ietf.org/rfc/rfc4627.txt.

[259] S. JOSEFSSON. The Base16, Base32, and Base64 data encodings. RFC 4648 (Proposed Standard), Oct. 2006. URL: http://www.ietf.org/rfc/rfc4648.txt.

[260] R. MOATS. URN syntax. RFC 2141 (Proposed Standard), May 1997. URL: http://www.ietf.org/rfc/rfc2141.txt.

[261] G. MOHR. MAGNET, June 2002. URL: http://magnet-uri.sourceforge.net/magnet-draft-overview.txt.

[262] M. MYERS, R. ANKNEY, A. MALPANI, S. GALPERIN, AND C. ADAMS. X.509 Internet public key infrastructure Online Certificate Status Protocol — OCSP. RFC 2560 (Proposed Standard), June 1999. Updated by RFC 6277. URL: http://www.ietf.org/rfc/rfc2560.txt.

[263] C. NEUMAN, T. YU, S. HARTMAN, AND K. RAEBURN. The Kerberos network authentication service (V5). IETF, July 2005.

[264] C. NEUMAN, T. YU, S. HARTMAN, AND K. RAEBURN. The Kerberos network authentication service (V5). RFC 4120 (Proposed Standard), July 2005. Updated by RFCs 4537, 5021, 5896, 6111, 6112, 6113. URL: http://www.ietf.org/rfc/rfc4120.txt.

[265] M. SIRBU. Content-type header field for Internet messages. RFC 1049 (Historic), Mar. 1988. URL: http://www.ietf.org/rfc/rfc1049.txt.

NEWS ORGANIZATIONS AND PRESS RELEASES

[266] U. S. to act to oust ship work slackers. *New York Tribune*, page 9, Sept. 1918. URL: http://chroniclingamerica.loc.gov/lccn/sn83030214/1918-09-22/ed-1/seq-9/.

[267] *Forbes*, 143:288, 1989. URL: http://books.google.co.uk/books?id=iOkdAQAAMAAJ.

[268] Public safety minister launches investigation into Facebook postings [online]. Oct. 2007. CBC News. URL: http://www.cbc.ca/news/canada/british-columbia/story/2007/10/02/bc-dayinvestigation.html.

[269] Student recruits unfit for service, say former border guards [online]. Oct. 2007. CBC News. URL: http://www.cbc.ca/canada/british-columbia/story/2007/10/01/bc-borderguards.html.

[270] Facebook agrees to address Privacy Commissioner's concerns, Aug. 2009. Office of the Privacy Commissioner of Canada. URL: http://www.priv.gc.ca/media/nr-c/2009/nr-c_090827_e.cfm.

[271] Facebook announces privacy improvements in response to recommendations by Canadian privacy commissioner, Aug. 2009. Facebook. URL: http://www.facebook.com/press/releases.php?p=118816.

[272] E. BARNETT. Steve Jobs: 'I admire Facebook's Mark Zuckerberg' [online]. Oct. 2011. The Telegraph. URL: http://www.telegraph.co.uk/technology/steve-jobs/8846021/Steve-Jobs-I-admire-Facebooks-Mark-Zuckerberg.html.

[273] J. CHENG. Over 3 years later, "deleted" Facebook photos are still online [online]. Feb. 2012. Ars Technica. URL: http://arstechnica.com/business/news/2012/02/nearly-3-years-later-deleted-facebook-photos-are-still-online.ars.

[274] S. DIAZ. Sponsored Stories: Facebook's effort to use your "likes" and more in ads [online]. Jan. 2011. ZDNet. URL:

http://www.zdnet.com/blog/btl/sponsored-stories-facebooks-effort-to-use-your-likes-and-more-in-ads/44014.

[275] D. GOLDMAN. Rapleaf is selling your identity [online]. Oct. 2010. CNN Money. URL: http://money.cnn.com/2010/10/21/technology/rapleaf/index.htm.

[276] D. GOODIN. Code for open-source Facebook littered with landmines [online]. Sept. 2010. The Register. URL: http://www.theregister.co.uk/2010/09/16/diaspora_pre_alpha_landmines.

[277] T. HODGKINSON. Facebook IPO: Log off! Facebook is ruthlessly selling your soul [online]. Feb. 2012. The Daily Mail. URL: http://www.dailymail.co.uk/news/article-2095690/Facebook-IPO-Log-Facebook-ruthlessly-selling-soul.html.

[278] J. A. KAPLAN. Facebook to sell YOUR posts to advertisers. *Fox News*, July 2012. URL: http://www.foxnews.com/tech/2011/01/26/facebook-friends-used-ads.

[279] L. MAGID. Facebook details new privacy settings [online]. Dec. 2009. CNet. URL: http://news.cnet.com/8301-19518_3-10411418-238.html?tag=mncol;txt.

[280] C. MCCARTHY. Facebook backtracks on public friend lists [online]. Dec. 2009. URL: http://news.cnet.com/8301-13577_3-10413835-36.html.

[281] J. C. PEREZ. Facebook will shut down Beacon to settle lawsuit [online]. Sept. 2009. PC World. URL: http://www.pcworld.com/article/172272/facebook_will_shut_down_beacon_to_settle_lawsuit.html.

[282] E. STEEL AND G. A. FOWLER. Facebook in privacy breach. *The Wall Street Journal*, Oct. 2010. URL: http://online.wsj.com/article/SB10001424052702304772804575558484075236968.html.

Facebook Blog

[283] Canvas encryption proposal [online]. Oct. 2010. URL: https://developers.facebook.com/docs/authentication/canvas/encryption_proposal/.

[284] C. Abram. Welcome to Facebook, everyone. [online]. Sept. 2006. URL: https://blog.facebook.com/blog.php?post=2210227130.

[285] E. Beard. A new data model [online]. Apr. 2010. URL: https://developers.facebook.com/blog/post/378.

[286] D. Fetterman. Facebook Development Platform launches... [online]. Aug. 2006. URL: https://blog.facebook.com/blog.php?post=2207512130.

[287] P. Fung. Public search listings on Facebook [online]. Sept. 2007. URL: https://blog.facebook.com/blog.php?post=2963412130.

[288] N. Gupta. Facebook Platform roadmap update [online]. Aug. 2010. URL: https://developers.facebook.com/blog/post/402.

[289] A. Haugen. New ways to find and engage with your favorite applications [online]. Oct. 2009. URL: https://blog.facebook.com/blog.php?post=166797817130.

[290] A. Haugen. Answers to your questions on personalized Web tools [online]. Apr. 2010. URL: https://blog.facebook.com/blog.php?post=384733792130.

[291] A. Haugen. New ways to personalize your online experience [online]. Apr. 2010. URL: https://blog.facebook.com/blog.php?post=383515372130.

[292] R. C. He. The Facebook open stream API [online]. Apr. 2009. URL: https://developers.facebook.com/blog/post/225.

[293] R. C. HE. New privacy controls for your applications [online]. Feb. 2010. URL: http://blog.facebook.com/blog.php?post=311056167130.

[294] A. LI. Connecting to everything you care about [online]. Apr. 2010. URL: http://blog.facebook.com/blog.php?post=382978412130.

[295] P. MCDONALD. Growing beyond regional networks [online]. June 2009. URL: https://blog.facebook.com/blog.php?post=91242982130.

[296] A. MULLER. Updates on your new privacy tools [online]. Dec. 2009. URL: https://blog.facebook.com/blog.php?post=197943902130.

[297] L. PEARLMAN. Facebook ads [online]. Nov. 2007. URL: https://blog.facebook.com/blog.php?post=6972252130.

[298] D. PURDY. Moving to a modern platform [online]. Sept. 2011. URL: https://developers.facebook.com/blog/post/568/.

[299] A. RICE. A continued commitment to security [online]. Jan. 2011. URL: http://blog.facebook.com/blog.php?post=486790652130.

[300] J. ROSENSTEIN. Opening up [online]. Nov. 2007. URL: https://blog.facebook.com/blog.php?post=7057627130.

[301] R. SANGHVI. Facebook gets a facelift [online]. Sept. 2006. URL: https://blog.facebook.com/blog.php?post=2207967130.

[302] R. SANGHVI. New tools to control your experience [online]. Dec. 2009. URL: https://blog.facebook.com/blog.php?post=196629387130.

[303] M. E. SHARON. Who, what, when, and now... where [online]. Aug. 2010. URL: https://blog.facebook.com/blog.php?post=418175202130.

[304] A. STEINBERG. FQL [online]. Feb. 2007. URL: https://blog.facebook.com/blog.php?post=2245872130.

[305] B. TAYLOR. Applications ask, you receive: simplified permissions launch [online]. June 2010. URL: https://blog.facebook.com/blog.php?post=403443752130.

[306] B. TAYLOR. Bringing your friends to Bing: search now more social [online]. Oct. 2010. URL: https://blog.facebook.com/blog.php?post=437112312130.

[307] M. VERNAL. An update on encrypted UIDs [online]. Nov. 2010. URL: https://developers.facebook.com/blog/post/431/.

[308] M. VERNAL. An update on Facebook UIDs [online]. Oct. 2010. URL: https://developers.facebook.com/blog/post/422/.

[309] M. VERNAL. Encrypting Facebook UIDs [online]. Oct. 2010. URL: https://developers.facebook.com/blog/post/419.

[310] M. ZUCKERBERG. An open letter from Mark Zuckerberg [online]. Sept. 2006. URL: https://blog.facebook.com/blog.php?post=2208562130.

[311] M. ZUCKERBERG. Calm down. Breathe. We hear you. [online]. Sept. 2006. URL: https://blog.facebook.com/blog.php?post=2208197130.

[312] M. ZUCKERBERG. Thoughts on Beacon [online]. Dec. 2007. URL: https://blog.facebook.com/blog.php?post=7584397130&fb_comment_id=fbc_7584397130_17166601_10150192637137131#us14yl_1.

[313] M. ZUCKERBERG. Building the social Web together [online]. Apr. 2010. URL: https://blog.facebook.com/blog.php?post=383404517130.

[314] M. ZUCKERBERG. Giving you more control [online]. Oct. 2010. URL: https://blog.facebook.com/blog.php?post=434691727130.

[315] M. ZUCKERBERG. Making control simple [online]. May 2010. URL: https://blog.facebook.com/blog.php?post=391922327130.

OTHER WEBSITES

[316] A penny per match [online]. URL: https://www.rapleaf.com/pricing/.

[317] Amazon S3 pricing [online]. URL: http://aws.amazon.com/s3/pricing/.

[318] Google Apps service level agreement [online]. URL: http://www.google.com/apps/intl/en/terms/sla.html.

[319] Rapleaf response for demo e-mail address [online]. URL: https://personalize.rapleaf.com/v4/dr?email=vlad@rapleafdemo.com&show_available&format=html&api_key=05664e263f1481a7e9ed1e8b85e2790e.

[320] Suzanne Massie during the Reagan years [online]. URL: http://www.suzannemassie.com/reaganYears.html.

[321] Amazon S3 service level agreement [online]. Oct. 2007. URL: http://aws.amazon.com/s3-sla/.

[322] *launchd(8)*, May 2009. URL: https://developer.apple.com/library/mac/#documentation/Darwin/Reference/Manpages/man8/launchd.8.html.

[323] About /robots.txt [online]. Aug. 2010. URL: http://www.robotstxt.org/robotstxt.html.

[324] Making search more secure [online]. Oct. 2011. URL: http://googleblog.blogspot.co.uk/2011/10/making-search-more-secure.html.

[325] Security overview. Technical report, July 2011. URL: http://developer.apple.com/library/mac/#DOCUMENTATION/Security/Conceptual/Security_Overview/Concepts/Concepts.html.

[326] Sponsored Stories guide. *ads.ak.facebook.com*, 2011. URL: http://ads.ak.facebook.com/ads/FacebookAds/Sponsored_Stories_Guide_US.pdf.

[327] ï¿Œï¿ŒMicrosoft Exchange online dedicated plans version service level agreement (SLA). pages 1–4, Oct. 2011.

[328] Amazon S3 price reduction [online]. Feb. 2012. URL: http://aws.typepad.com/aws/2012/02/amazon-s3-price-reduction.html.

[329] *android.app.Service*, api level 15 edition, June 2012.

[330] Cloud Files pricing - online storage & CDN media delivery [online]. 2012. URL: http://www.rackspace.com/cloud/cloud_hosting_products/files/pricing/.

[331] Connect with Facebook [online]. 2012. URL: http://www.spotify.com/uk/about/features/connect-with-facebook/.

[332] Draw Something [online]. 2012. URL: http://omgpop.com/drawsomething/.

[333] Facebook's filing: the highlights [online]. Feb. 2012. URL: http://bits.blogs.nytimes.com/2012/02/01/facebooks-filing-the-highlights/.

[334] FarmVille [online]. 2012. URL: http://company.zynga.com/games/farmville.

[335] *Getting Tahoe-LAFS*. trunk/docs/quickstart.rst, r5548 edition, June 2012. URL: https://tahoe-lafs.org/trac/tahoe-lafs/browser/trunk/docs/quickstart.rst.

[336] Google App Engine pricing [online]. 2012.

[337] Pricing [online]. 2012. URL: http://www.netdna.com/pricing/.

[338] Securing your Twitter experience with HTTPS [online]. Feb. 2012. URL: http://blog.twitter.com/2012/02/securing-your-twitter-experience-with.html.

[339] K. Atkinson. Spell checker oriented word lists (SCOWL), Jan. 2011. URL: http://wordlist.sourceforge.net/.

[340] R. Atterer. jigdo 0.6.1 - please test! [online]. Dec. 2001. URL: http://lists.debian.org/debian-cd/2001/12/msg00059.html.

[341] E. Bangeman. Blu-ray's DRM crown jewel tarnished with crack of BD+ [online]. Nov. 2007. URL: http://arstechnica.com/uncategorized/2007/11/blu-rays-drm-crown-jewel-tarnished-with-crack-of-bd/.

[342] BLUE BEETLE. User-driven discontent [online]. Aug. 2010. URL: `http://www.metafilter.com/95152/Userdriven-discontent#3256046`.

[343] J. BONNEAU. How privacy fails: the Facebook applications debacle [online]. June 2009. URL: `http://www.lightbluetouchpaper.org/2009/06/09/how-privacy-fails-the-facebook-applications-debacle/`.

[344] A. CHAUDHRY, A. MADHAVAPEDDY, C. ROSTOS, R. MORTIER, A. AUCINAS, J. CROWCROFT, S. P. EIDE, S. HAND, A. W. MOORE, AND N. VALLINA-RODRIGUEZ. Signposts. *github.com*, 2011. URL: `https://github.com/avsm/signpost`.

[345] A. CHAUDHRY, A. MADHAVAPEDDY, C. ROSTOS, R. MORTIER, A. AUCINAS, J. CROWCROFT, S. P. EIDE, S. HAND, A. W. MOORE, AND N. VALLINA-RODRIGUEZ. Signposts: end-to-end networking in a world of middleboxes. SIGCOMM 2012 Poster & Demo Session, Aug. 2012. URL: `http://conferences.sigcomm.org/sigcomm/2012/program.php`.

[346] C. FLECK. Google+ = work friends, Facebook = family, Linkedin = business contacts, Twitter = social bookmarks [online]. July 2011. URL: `http://blogs.citrix.com/2011/07/17/google-work-friends-facebook-family-linkedin-business-contacts-twitter-social-bookmarks/`.

[347] P. GANAPATI. Scribd Facebook instant personalization is a privacy nightmare [online]. Sept. 2010. URL: `http://www.wired.com/epicenter/2010/09/scribd-facebook-instant-personalization/`.

[348] C. LEE. *The inetd super-server*. URL: `http://www.freebsd.org/doc/handbook/network-inetd.html`.

[349] J. LINFORD. 5 things to consider before taking up social media [online]. URL: `http://www.webpresence.tv/uk-blog/5-ways-social-media-wont-fix-website-2/`.

[350] C. MATYSZCZYK. Zuckerberg: I know that people don't want privacy [online]. Jan. 2010. URL: `http://news.cnet.com/8301-17852_3-10431741-71.html`.

[351] M. S. Miller, M. Samuel, B. Laurie, I. Awad, and M. Stay. Caja: safe active content in sanitized JavaScript [online]. Jan. 2008. URL: `http://google-caja.googlecode.com/files/caja-spec-2008-01-15.pdf`.

[352] A. Newitz. Defenses lacking at social network sites [online]. Dec. 2003. URL: `http://www.securityfocus.com/news/7739`.

[353] A. Peslyak. Getting around non-executable stack (and fix) [online]. Aug. 1997. URL: `http://insecure.org/sploits/linux.libc.return.lpr.sploit.html`.

[354] S. Prather. Fire, ready, aim! [online]. July 2006. URL: `http://peoplegen.blogspot.co.uk/2006/07/fire-ready-aim.html`.

[355] J. Reimer. New AACS cracks cannot be revoked, says hacker [online]. Apr. 2007. URL: `http://arstechnica.com/gadgets/2007/04/aacs-cracks-cannot-be-revoked-says-hacker/`.

[356] S. Schillace. Default https access for Gmail [online]. Jan. 2010. URL: `http://gmailblog.blogspot.co.uk/2010/01/default-https-access-for-gmail.html`.

[357] F. A. Stevenson. Cryptanalysis of Contents Scrambling System [online]. Nov. 1999. URL: `http://www.dvd-copy.com/news/cryptanalysis_of_contents_scrambling_system.htm`.

[358] Z. Wilcox-O'Hearn. convergent encryption reconsidered [online]. Mar. 2008.