Secure Public Instant Messaging

By

Mohammad Abdul Mannan

A thesis submitted to

the Faculty of Graduate Studies and Research

in partial fulfilment of

the requirements for the degree of

Master of Computer Science

Ottawa-Carleton Institute for Computer Science

School of Computer Science

Carleton University

Ottawa, Ontario, Canada

August 2005

©Copyright

Mohammad Abdul Mannan, 2005

The undersigned hereby recommend to

the Faculty of Graduate Studies and Research

acceptance of the thesis,

Secure Public Instant Messaging

submitted by

Mohammad Abdul Mannan

Dr. Douglas Howe

(Director, School of Computer Science)

Dr. Paul Van Oorschot

(Thesis Supervisor)

Carleton University

August 2005

Abstract

Although Instant Messaging (IM) services are quite mature and very popular as an instant way of communication over the Internet, they have received barely any attention from the security research community. We provide a survey on security features and threats to existing IM networks and discuss how currently available systems fail to provide adequate security in light of existing threats.

Despite important differences distinguishing IM from other Internet applications, no protocols have been designed to adequately deal with the unique security issues of IM. We present the Instant Messaging Key Exchange (IMKE) protocol as a step towards secure IM. IMKE is designed to provide security in the present Internet threat model. It is intended to be embedded in (as a small change to) popular IM protocols, not to function as another independent messaging protocol. A discussion of realistic threats to IM and a related analysis of IMKE using a BAN (Burrows-Abadi-Needham)-like [30] logic is also provided. An implementation of IMKE using the open-source Jabber protocol is provided as well.

Acknowledgments

I gratefully acknowledge the insightful feedback, constructive suggestions, and corrections put forth by Paul C. van Oorschot, Liam Peyton and Anil Somayaji. As well, I thank Pat Morin for chairing the defence of this thesis. Thanks to all members of Carleton's Digital Security Group for their enthusiastic discussions on this popular topic, especially Glenn Wurster, David Whyte, and Julie Thorpe.

Contents

Abstract **i**

Acknowledgments **ii**

1 Introduction and Overview of Instant Messaging **1**

1.1 Introduction 1

1.2 Basics of IM Protocols and Features 7

2 Related Work on IM and Password Authentication **11**

2.1 IM and IM Security 11

2.2 Existing Security Mechanisms 15

2.2.1 Security and Privacy Features in Default Clients 15

2.2.2 Third-Party Solutions 16

2.3 Comparison of IMKE with other IM Implementations 21

2.4 Password Authentication 23

3 Security Threats to IM **26**

3.1 General Threats 26

3.2 Threats Resulting from IM Features 29

4 IM Worms, Analysis and Countermeasures **34**

4.1 Analysis of Selected IM Worms and Vulnerabilities 35
4.2 Distinguishing Features of IM Networks 39
4.3 Topology of the Network formed by IM Contacts 41
4.4 Measures for Limiting IM Worm Epidemics 43
4.4.1 Existing Techniques and Remarks Thereon 43
4.4.2 New Proposals 47
4.4.3 User Study on Per-User Frequency of IM Text Messaging and File Transfer 50

5 IMKE: Instant Messaging Key Exchange 53
5.1 Motivations for IMKE 54
5.1.1 Relationship of IMKE to Two- and Three-Party Protocols . . 55
5.1.2 Relationship of IMKE to EKE and Similar Protocols 56
5.2 Setup for IMKE 59
5.2.1 Threats Considered in IMKE 59
5.2.2 Notation, Goals and Secrets 61
5.3 The IMKE Protocol 62
5.3.1 Password Authentication and Key Exchange (PAKE) 64
5.3.2 Client-Server Communications 67
5.3.3 Client-Client Communications (Direct and Relayed) 68

6 Security and Performance Analysis of IMKE 72
6.1 Security Analysis 72
6.1.1 Setup for the Analysis 73
6.1.2 Analysis of IMKE Messages 75
6.1.3 Other Security Attributes of IMKE 80
6.2 Performance Analysis (Analytical) 84

7 Implementation of IMKE **87**

7.1 Protocol Specification 88

7.2 IMKE Authentication Message Flow in Jabber 92

7.3 Empirical Performance, Usability, and Deployment 94

7.4 Lessons Learned 99

8 Conclusions and Future Work **101**

List of Tables

1.1 IM-related definitions 8

2.1 Comparison of IM implementations 22

4.1 Average file transfer, text messages, and online users over 15-minute intervals 51

4.2 Comparison of file transfer (FT) and text message (TM) usage . . . 51

5.1 Threats to IM and those addressed by IMKE 59

5.2 End-user goals in IMKE 62

5.3 Notation and terminology used in IMKE 63

6.1 BAN-like definitions used in the IMKE analysis 73

6.2 Technical sub-goals of IMKE 74

6.3 Operational assumptions of IMKE 75

6.4 Summary of IMKE messages (see Table 5.3 for notation) 76

6.5 IM threat model assumptions 77

6.6 Cryptographic operations required by IMKE in the PAKE phase . . . 84

7.1 Cryptosystems and parameters for the IMKE implementation 89

7.2 Test setup: machine configuration 95

7.3 Summary of IMKE messages (repeated) 96

7.4 Message execution time (in milliseconds) of IMKE (test client) 97

7.5 Division of the PAKE phase execution time (test client) 97

7.6 Comparison of the XMPP and IMKE Gaim implementations 97

List of Figures

1.1 IM communications models 10

2.1 Weaknesses of the IMSecure model 19

4.1 Throttle algorithm for IM [184] 45

Chapter 1

Introduction and Overview of Instant Messaging

This chapter provides a brief introduction to Instant Messaging (IM), outlines what motivates our research, scope, contributions and how the rest of this thesis is organized. An overview of IM protocols and features is also given.

1.1 Introduction

IM is a communication service over the Internet that enables individuals to exchange text messages and track the availability of a list of users in near real-time. IM systems have roots in UNIX applications (e.g. `talk` and `write`), in which users on the same server can exchange text-messages in a conversation mode but cannot track availability. IM usage gradually increased with the early implementations of the MIT Project Athena Zephyr notification system [36], Internet Relay Chat (IRC, started at the University of Oulu in Finland; see [123, 79]), and the introduction of the *buddy list* feature to track users' availability in AOL Instant Messenger (AIM) [3]. How-

ever, the recent rise in popularity of consumer IM services has been phenomenal (e.g. see [89]). Starting as a casual application, mainly used by young adults and college students, IM systems now connect even naval operations (over 300 US Navy warships are connected via an IM service) and various customer services [33].

There are many public domain IM services. The most popular include AIM [3], ICQ [66], MSN Messenger (Windows Messenger in Windows XP) [102], and Yahoo! Messenger (YIM) [188]. We focus on these messaging networks and their default clients.[1] There are also many third-party[2] clients that interact on these networks. We discuss both the third-party and default clients in terms of the security risks associated with them. The basic protocols currently used in public IM systems are open to many security threats (see Chapter 3 and 4). Security techniques, e.g. TLS/SSL connections or digital certificates, used in corporate IM systems are inadequate to address these threats (see below).

Motivation.

IM differs from many other Internet applications because of its near real-time nature of user interactions, e.g. online presence notification and instant messages. Consequently, many security mechanisms designed for other Internet applications (e.g. web browser, email) are inadequate for IM. Despite the immense popularity of IM systems (both in the consumer and business world), security issues related to IM have largely been ignored by the security research community. To our knowledge, there exists no complete security protocol suite in the literature specifically tailored for password-based IM systems.

[1] By *default clients* we mean the IM clients provided by public IM service providers (e.g. MSN Messenger).

[2] By *third-party* clients we refer to clients (e.g. Gaim [125], Trillian [32], IMSecure [192]) which interact with the existing major IM networks, and security-enhanced IM products (e.g. Yahoo! Business Messenger [187]).

The current Internet Threat Model [139, p.1] (including the SSL model) assumes a vulnerable communication link with trusted end-points. However, the assumption of secure end-points may undermine software security, as the present Internet environment is infested with malicious software compromising a large number of machines at any given point of time. A Sept. 2004 survey [118] projected that about 91 percent of PCs are infected with spyware programs (see also [151, 150]); so the assumption of secure end-points is no longer a useful practical model for general Internet users. Also, the SSL model secures connections between two entities, whereas most IM connections involve three parties – two users and one server. Because of this limited threat model, SSL-based solutions appear inadequate for securing IM. IM security is discussed in this thesis with respect to an *extended* threat model (see Section 6.1.1) which takes (not necessarily trusted) end-points into consideration.

Some of the security threats to IM are similar to those for email, for example, misleading web links (often used to "phish" [34] for passwords and other tokens for identity theft) and malcode execution from received files. Anti-virus tools to protect email from such threats are quite mature and relatively effective. For email, such tools can be implemented at the gateway level, as monitoring email traffic is essentially straightforward. For IM, the use of non-standard proprietary protocols and decentralized peer-to-peer (P2P) file transfer makes it difficult to monitor IM traffic at the gateway level. Hence, incorporating similar protection mechanisms as used for email appears to be more difficult, and they provide at best limited protection against IM threats.

We highlight threats to IM to create greater public awareness of the danger of using present IM systems, and to improve security in the long term (although in the short term this may increase the risk of these threats becoming reality). We seek to help lay the foundation to advance research in the area of secure IM, as a first step

towards improving security of IM systems.

Contributions.

We provide a comprehensive survey [99] of threats to IM, especially the threats posed by IM worms. Existing security mechanisms for IM as well as strategies to contain IM worms are discussed. We propose a new protocol, the Instant Messaging Key Exchange (IMKE) protocol, for securing public IM, and techniques to restrict IM worms' propagation [100]. IMKE enables mutual *strong* authentication (for definition see Table 5.3) between users and an IM server, using a memorable password (e.g. like EKE [16]) and a known server's public key. IMKE provides security (authentication, confidentiality and integrity) for client-server and client-client IM connections with *repudiation* (for definition see Table 5.3). Although pairs of users generally share no secret between themselves, IMKE enables secure and private communications among users through a trusted IM server, without revealing the contents of users' messages to the server.

An analysis of the protocol in terms of security and performance is provided (although it is not a full proof of the security of IMKE). We also discuss how IMKE avoids some recently devised attacks on Password Authentication and Key Exchange (PAKE) protocols in addition to classic ones. IMKE may be implemented using any well-known public key cryptosystem (e.g. RSA, ElGamal, Rabin) that supports encryption. In contrast, the majority of the proposed PAKE protocols are based on the Diffie-Hellman (DH)-based key agreement; nevertheless, there are known attacks which exploit the structure of the parameters in the DH-based key agreement (e.g. [28, 88, 178, 90, 4, 191]). For a secure implementation of IMKE, general requirements for secure choice of public key parameters must be fulfilled (e.g. see [4]); however,

we argue that IMKE may not require any additional special constraints (unlike e.g. SNAPI [96]) for a safe protocol run.

We have implemented a prototype of IMKE as a part of this research using the Jabber open-source IM protocol [148, 149]. Although implementing IMKE requires changing both the IM server and the IM client, we show how IMKE may be integrated with public IM protocols without requiring a large implementation effort.

Although IMKE has been designed as a secure protocol specifically for IM, it may provide an alternative to the patented EKE, and other two- and three-party strong PAKE protocols beyond IM; a major architectural difference is that we use the known public key of the IM server.

Scope.

Some vendors provide IM services for mobile devices. The Short Messaging System (SMS) was created as part of the Global System for Mobile (GSM) Communications Phase 1 standard. IM in mobile devices, SMS, IRC, group chat, and chat rooms (see Section 1.2 for definitions) are beyond the scope of this thesis.

IM systems with message logging on the server side is a required feature at some organizations (e.g. financial firms for regulatory reasons, such as the Sarbanes-Oxley Act [176]). The idea of *mobile* IM, introduced by Issacs *et al.* [71], is establishing a foothold on major messaging systems; AIM supports login from multiple devices at a time to enhance user mobility. These features, although useful, are out of scope of this thesis.

Our main focus is the (one-to-one) PC-to-PC messaging, which is the dominating feature of all IM systems. IM services mainly targeting corporate users, such as Yahoo! Business Messenger, are not fully analyzed in this thesis (in part because complete documentation of security features in these products is not publicly available). As the

default IM clients discussed here are mainly Windows based, Windows is generally implied to be the underlying operating system (OS) when another is not explicitly mentioned.

Security and privacy issues related to IM can be categorized as technical and social. Technical threats arise from inherent system design and implementation bugs. Social issues include: divulging sensitive information to strangers or competitors using IM, impacts of IM on personal relationship and workplace, etc. We deal only with the technical issues of IM (keeping usability in context).

Outline.

The rest of the thesis is organized as follows. Section 1.2 summarizes the basic protocols used in mainstream IM systems. Chapter 2 briefly reviews related work regarding IM security and well-known PAKE protocols. Privacy and security features of current IM services and third-party IM security solutions are also briefly discussed in Chapter 2, along with weaknesses of third-party solutions. A comparison of IMKE with other *secure* IM implementations is also provided. Chapter 3 briefly discusses the most significant security threats to IM. Threats from IM worms are discussed in Chapter 4, where we also provide an outline of the topology of IM contacts' networks for better understanding of IM worms' characteristics, and introduce techniques complementary to IMKE to restrict IM worms' propagation. We propose the IMKE protocol for authentication and communications (client-server and client-client) to secure IM network connections in Chapter 5. Chapter 6 provides our IM threat model and a security and (analytical) performance analysis of IMKE. Implementation details of IMKE in Jabber are provided in Chapter 7. Chapter 8 discusses future work and conclusions of this thesis.

1.2 Basics of IM Protocols and Features

This section provides a review of public IM protocols, communication models, and common IM features. To facilitate further discussion, we provide a few IM-related definitions in Table 1.1.

Descriptions of the protocols used by the major IM networks are available on the Internet. The remainder of this section contains a brief architectural overview of popular IM networks. Reverse-engineered details of the AIM, YIM and MSN protocols are available on many web sites (e.g. [48, 179, 108]).

Common features supported in most IM clients include: contact lists; block lists; instant text messages (one-to-one, multi-user); presence information; availability (available, away, busy etc.); email; sending and receiving files, URLs; audio and video chat; sharing external applications (e.g. web browser); online games; setting permission levels for different types of users (e.g. contact list, everyone); and message archiving.

Most communications in IM systems are client-server based, where each user shares a secret, user-chosen (often *weak*) password with the IM server. A client normally sends the password hash to the server for authentication. Messages among users are also typically relayed through the server (mainly to avoid firewall issues). However, purely P2P communications also occur in some situations (e.g. audio/video chat, file transfer). Most IM communications occur over TCP; however, UDP is sometimes used for P2P connections. Also, SSL is used in some corporate IM services (e.g. Reuters Messaging [140]) and during the authentication phase of the currently available MSN protocol.

Different widely used IM clients cannot communicate with each other, mainly because of their proprietary and incompatible protocols. The Jabber IM protocol has

online user	A user successfully logged in to an IM server.
presence	Presence information reveals whether or not a user is logged in to an IM server.
availability/user mode	Availability information reveals a user's willingness (e.g. "busy", "do not disturb") to send/receive messages, or status (e.g. "away", "on the phone").
contact/buddy list	The list of user IDs whose presence and availability a user has currently subscribed to.
block list	The list of user IDs explicitly barred from getting the current user's presence and availability information; listed users also cannot send any messages to the current user.
allow list	The list of user IDs allowed to send messages to the current user and which can track the user's presence and availability information.
one-to-one chat	A user sends or receives messages from another user, generally through the IM server.
group chat	More than two users exchange messages at a time. Users form a virtual "group", generally which is short-lived. Users in a group chat are usually closely related.
chat room	A virtual room, generally consisting of many users who exchange instant messages on some closely related topics.
IM user	A human user of an IM service.
IM client	A (software) program that enables a human user to use an IM service.
IM server	A (software) program that enables IM clients to access IM features in an IM service.
IRC	A client-server based multi-user messaging system of large networks. Users meet on "channels" (rooms/virtual places, usually with a certain topic of conversation) for group or personal messaging.

Table 1.1: IM-related definitions

been designed so that IM servers run by different organizations can communicate. The Jabber Software Foundation[3] initially developed the Jabber protocol which has evolved in open-source communities. The Internet Engineering Task Force[4] (IETF) has approved the base Jabber protocol – also known as the Extensible Messaging and Presence Protocol (XMPP) [148, 149] – as a standard IM protocol. Jabber is based on a set of XML streaming protocols and uses a distributed client-server architecture like email. Anyone can run a Jabber server which can communicate with other Jabber servers using the standard Jabber protocol. Therefore, Jabber users can communicate with any other online Jabber user irrespective of the servers' location or management.

While an IM server appears to be a single entity to a client, it may be a group of servers controlled by a single IM service provider (e.g. AOL), or a collection of servers from independent IM service providers (e.g. Jabber). If user A wants to communicate with user B, both must log into the same IM service. Messages from A to B will be delivered by the server depending on B's privacy settings. For direct communications between A and B, the server provides the necessary information (e.g. network address) to each party. Figure 1.1 shows the standard IM communications model for single (centralized) and multiple (distributed) servers.

[3]http://www.jabber.org/
[4]http://www.ietf.org/

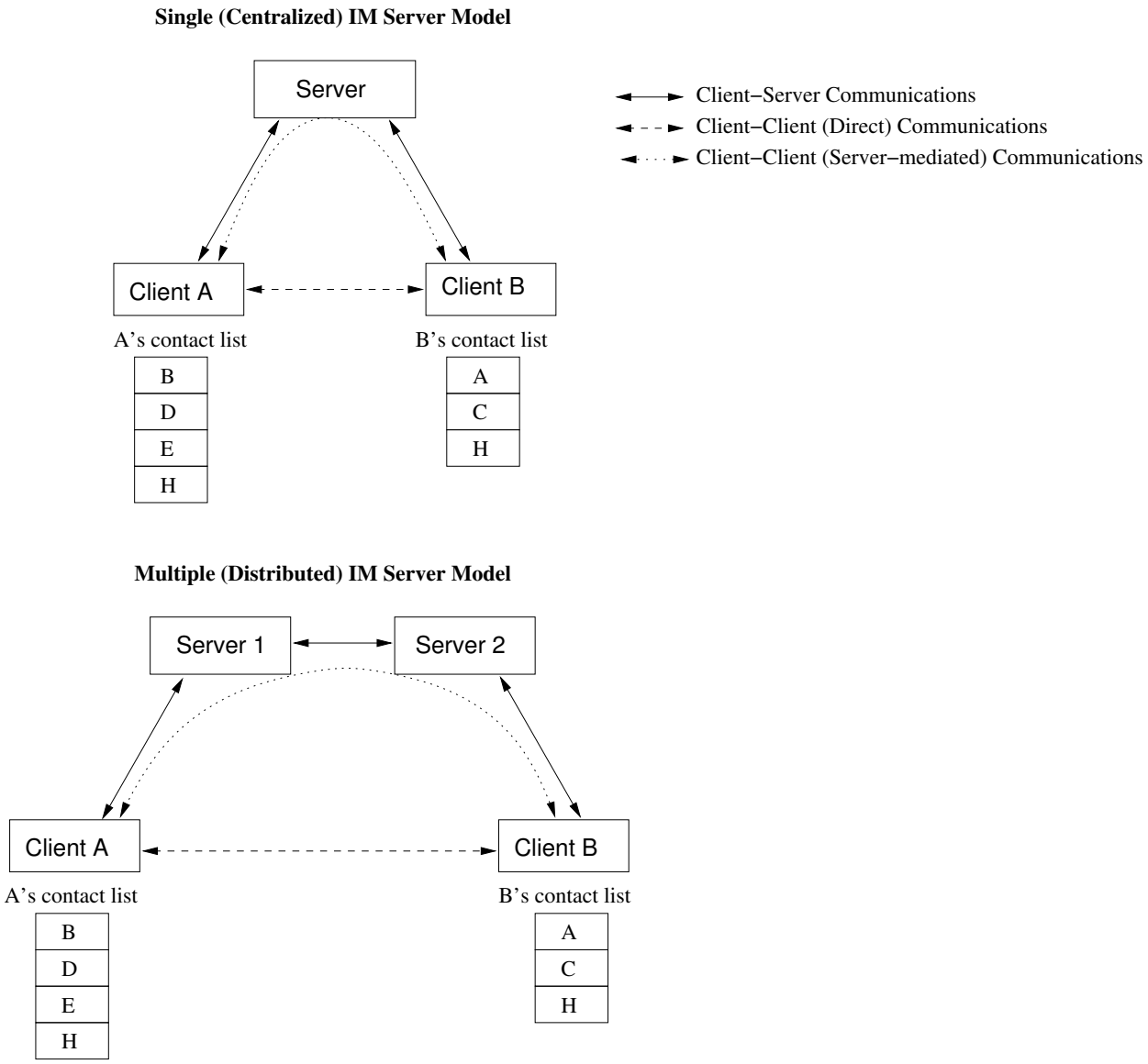

Figure 1.1: IM communications models

Chapter 2

Related Work on IM and Password Authentication

This chapter is a survey of the existing literature on IM and well-known IM security products. However, instead of Human Computer Interaction (HCI) or feature related issues, we focus on security risks of IM. As IM is a password-based system, and our proposed Instant Messaging Key Exchange (IMKE) protocol (see Chapter 5) is essentially a Password Authentication and Key Exchange (PAKE) protocol in part, we provide an overview of well-known PAKE protocols. A comparison of security and usability features of IMKE with selected *secure* IM implementations is also provided.

2.1 IM and IM Security

Much work (albeit mostly unrelated to security) has been done on IM and presence awareness systems in academia, mainly by HCI and Computer Supported Cooperative Work (CSCW) research groups. Several IM applications – e.g. Hubbub [72] (a sound-enhanced IM), KIM (Kinetic typography-based IM) [19], IMVis [114] (which

uses pictures and video snapshots to visualize contacts), and Threaded Chat [154] – have been designed to augment functionalities and to analyze usage. The Unified Messaging System [181] emerged from the pervasive computing idea that combines email, IM, newsgroups, SMS, paging etc. into one system. Many researchers have explored the effects of IM in the workplace. A study by Issacs et al. [73] found that 62% of IM conversations in the workplace were work related. Handel and Herbsleb [57] reported similar results (69% of recorded instant messages were work related). These results suggest positive contributions of IM in the workplace, although other researchers (e.g. [59, 182]) have expressed concerns of IM being used as a tool for *gossiping* or *goofing off*.

Ghavam et al. [49] presented approaches to integrating ad hoc communications into an enterprise framework through the use of secure group services and presence information. The goal of their framework is to enable and manage automatic and pervasive access to a set of secure communication services among users, by using Common Open Policy Service (COPS), Session Initiation Protocol (SIP) [147], and the TLS protocol. Godefroid et al. [51] proposed a framework that uses an automated verification approach to ensure conformance of complex dynamic presence awareness policies.

Related to IM security, a modified Diffie-Hellman protocol suitable to IM has been designed by Kikuchi et al. [82], primarily intended to secure message confidentiality against IM servers. It does not deal with the client-server authentication and also has limitations similar to the `IMSecure` [192] solution as discussed in Section 2.2. The security company Symantec provides reports (e.g. [61, 60]) on IM protocols, worms, threats and firewall issues. A web resource on the security analysis of Cerulean Studios' Trillian application is also available [110]. Informal discussions of security problems related to public IM in the enterprise environment are also available (e.g.

see Frase [47] re: some solutions using well-defined security policies and anti-virus tools).

IETF Efforts. IM protocol standardization efforts are ongoing in the IETF community in three main working groups: Instant Messaging and Presence Protocol (IMPP),[1] SIP for Instant Messaging and Presence Leveraging Extensions (SIMPLE) [31], and Extensible Messaging and Presence Protocol (XMPP, based on the Jabber IM protocol) [148, 149]. Several Internet-Drafts and Request For Comments (RFC) have been compiled by these groups. A comparative study on IM protocols including SIMPLE and XMPP is available due to Debbabi and Rahman [35]. Also, a Jabber white paper [74] compares the SIMPLE and XMPP protocols.

XMPP includes a method to protect an XML stream[2] from tampering and eavesdropping. XMPP can use the TLS protocol for stream encryption, along with a `STARTTLS` extension modeled after similar extensions proposed for the IMAP and POP3 protocols as described in RFC 2595 [116]. The Simple Authentication and Security Layer (SASL [115]) is proposed as a method for adding authentication support in XMPP. Note that, we use the terms XMPP and Jabber interchangeably throughout this thesis.

Security protocols and mechanisms for SIP are quite standardized. However, no specific security protocols have been developed focusing on SIMPLE. Mechanisms for authentication, end-to-end protection, replay and denial of service (DoS) attack prevention for SIMPLE rely on TLS and S/MIME (Secure/Multipurpose Internet Mail Extensions) protocols. Details of these security mechanisms are described in several RFCs (e.g. [31, 147, 146]).

An IETF proposed IM protocol standard, if adopted by the major IM vendors, will allow *interoperability* between IM services – i.e. users from different IM services will be

[1] http://www.ietf.org/proceedings/03mar/111.htm
[2] A container for the exchange of XML elements between any two entities over a network.

able to communicate using a common messaging platform. So far, such an adoption is far from reality, mainly due to financial implications of the *network effects*[3] of IM [46].

Off-the-Record Messaging. Following the real world scenario of off-the-record (OTR) conversations (e.g. only two persons talking in a closed room without any hidden tape recorder), Borisov et al. [22] proposed the "off-the-record communication" in 2004. The OTR communication model is designed to achieve *perfect forward secrecy* [133] and *repudiability*. (A protocol for off-the-record email [58] was introduced and implemented by AT&T in 2001.) For IM, Borisov et al. designed the "Off-the-record messaging" protocol which provides authentication and confidentiality of messages, and claims that when an IM conversation is over, no one, including the communicating parties, can produce a record of the messages exchanged. Users authenticate themselves by their known (published beforehand), long-term public keys (for digital signatures only). The Diffie-Hellman (DH) key agreement protocol is used to establish an encryption key and a MAC (Message Authentication Code) key. Every message is encrypted with a different encryption key. HMAC [84] is used to provide message authentication. Users only sign their initial DH public keys. This protocol has been implemented for the open-source `Gaim` [125] IM client as a plug-in. Certain aspects of this protocol result in characteristics which may be perceived as shortcomings. These include:

1. It requires users to possess long-term signature keys. For casual IM users, understanding and maintaining signature keys may not be practical (see Section 2.2.2 for more impacts of having a long-term public key).

[3] A service in which the value of a customer depends on the number of total customers using that service.

2. As the encryption and MAC key negotiation is a continuous process for the OTR messaging protocol, it may not be suitable for bulk data transfer between IM users (e.g. file transfer, audio/video chat).
3. Determined users can save the text messages they receive in OTR, and thereby keep a record of past conversations. Users may modify the available source code to keep track of the encrypted messages along with the stream of generated encryption keys; or, plaintext messages may be recorded when they are displayed in users' IM clients.

2.2 Existing Security Mechanisms

In this section, we list available security and privacy techniques in popular (default) IM clients, and summarize security features of well-known third-party IM solutions. Limitations of third-party and corporate IM solutions are also briefly discussed.

2.2.1 Security and Privacy Features in Default Clients

Recent versions of all major IM clients include an option to employ anti-virus software which can be launched automatically on every IM file download. An authorization option can be turned on so that explicit consent (though not cryptographically protected) of user A is required before A can be added to another user B's contact list. The same option is available for selecting who (users from A's contact list or everyone) can see user A's online status, and who can send messages and files to A. However, as "add-contact request" and "response" messages are transferred without any cryptographic protection, these messages are easy to spoof. In ICQ, a user can choose an option to select specific contacts who can see his/her online status even when the user is in *invisible* (logged in, but appearing offline to others) mode. ICQ

and YIM clients are equipped with word filters to replace or remove unwanted words from incoming text messages.

An ICQ client may ask a user to enter the login password every time the user wants to modify user-details, security and privacy permissions, and preferences settings. However, users need to turn on this feature explicitly after installation. ICQ has an option to accept/decline text-messages with URLs from everyone or only from those in a user's contact list.

IM clients are generally notified (by their IM server) when a new version of the client software is available with new features or security fixes, but users may choose not to upgrade, and generally software vendors allow older versions for backward compatibility. However, in October 2003, Microsoft introduced SSL-based authentication for MSN Messenger and disallowed login unless users updated their clients to the latest version. Another useful feature that all major IM vendors now provide is the protection against automated account creation by using CAPTCHA (completely automated public Turing test to tell computers and humans apart) [169]. This prevents software bots (i.e. automated programs) from signing up for an unlimited number of accounts, e.g. for use in sending unwanted messages to legitimate IM users.

2.2.2 Third-Party Solutions

Several IM products claim to be *secure*, although there is a lack of documentation about what is protected and what is not in these products. Our discussion is mainly based on available web resources, users' guides and help files (from software installation). The solutions are divided into the following categories: (1) SSL/TLS-based enterprise products; (2) anti-virus, firewall and IM gateway solutions; (3) public key based client-only solutions; and (4) independent secure IM protocols. Examples of

these categories are given below. We also briefly discuss security limitations of these products.

SSL/TLS-based Enterprise Products.

Yahoo! Business Messenger and Reuters Messaging protect instant messages using 128-bit SSL encryption. Using SSL-based solutions for public IM service, while a step forward in terms of security, has three major drawbacks:

1. limited threat model (see Section 1.1);
2. overhead for deployment at server-side (protocol is resource intensive and slow); and
3. messages may not be *private* when they go through a server, i.e. the server may view any encrypted message [82].

However, this last characteristic is desirable for message logging, albeit not when users value privacy to the extent that they prefer not to disclose their conversations to service providers.

Anti-virus, Firewall and IM Gateway Solutions.

`Norton AntiVirus` [159] comes with an IM plug-in for automatic scanning of incoming files. Also common anti-virus software can check any executable file before launching. However, anti-virus protections for IM currently guard against only known (or otherwise *detectable*) malware in file transfers. They cannot provide message (text or data) security (authentication, confidentiality, integrity) or protect against URL exploitations used for phishing.

The `ZoneAlarm` [193] personal firewall has a feature called *ID Lock*. This feature only protects user-configurable sensitive information like bank and credit card

numbers, home address, SIN (Social Insurance Number) etc. from being divulged in instant-message or email texts.

`IM Manager` [68] is an application-level proxy server for managing IM usage in an enterprise network. It also scans IM messages for malicious URLs and file transfer requests.

Public Key Based Client-only Solutions.

Here we analyze selected solutions that require installing security components (client software plug-ins) complementary to the default IM clients.

AIM clients can use a personal digital certificate to enhance authentication, integrity and confidentiality of text messaging.[4] GPG [127]-based `Gaim-e` [126] is another encryption plug-in for the popular cross-platform, multi-protocol (e.g. MSN, AOL, Yahoo!, Jabber) Gaim [125] IM client; the existence of long-term GPG public keys (distributed a priori) is assumed. Although solutions based on digital certificates (or trusted public keys) provide a high level of security, these solutions may be viewed as expensive (e.g. buying a digital certificate from VeriSign) for public domain users, and typically put the burden of certificate distribution, verification, expiry, renewal, revocation etc. on end-users. Furthermore, these solutions restrict users' mobility because users are required to *carry* the long-term certificates (and corresponding private keys).

`IMSecure` [192] provides encryption for popular IM services (e.g. AIM, MSN, Yahoo!). It may be installed with the default IM clients. During first-login, IMsecure generates a self-signed digital certificate for each of the user's IM accounts. Certificates are exchanged between two IMSecure clients while the users initiate an IM

[4] http://www.verisign.com/support/class1/secureaol.html

conversation. Public keys from digital certificates are used to establish the encryption key for an IM session. `Trillian` [32] provides similar security for AIM/ICQ users.

IMSecure and Trillian's solutions provide integrity and confidentiality, but not authentication. In these systems, instant messages are confidential between two users — in the sense that decrypting messages intercepted during transmission is computationally infeasible. However, such systems provide no protection against malware implanted in users' systems actively interacting with IM connections. An attack scenario is depicted in Figure 2.1. Also, control communications from IM clients to the IM server are not encrypted, even in the case of security sensitive information exchanged between clients and the server (e.g. contact list, presence, availability). Another disadvantage of IMSecure is that it must be installed (along with the default IM clients) in each system that a user wants to use for secure messaging as well as the intended recipients.

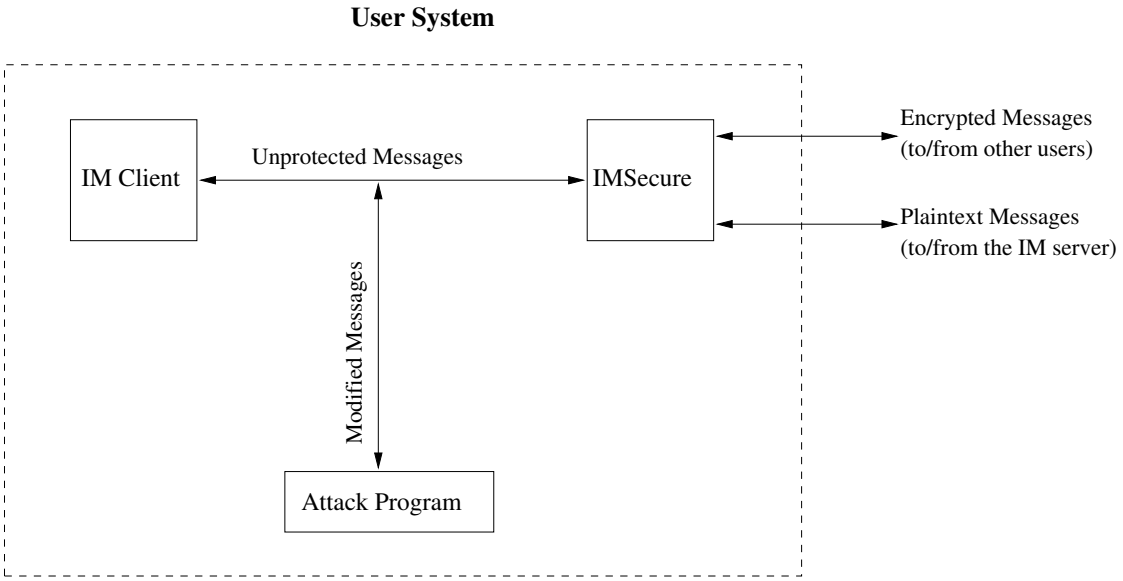

Figure 2.1: Weaknesses of the IMSecure model

Independent Secure IM Protocols.

Only a few secure IM protocols have been developed in practice. However, these protocols do not appear to have been peer-reviewed in an academic sense. They also require a new IM client and server – i.e. they are not designed to be integrated with popular IM protocols.

The Secure Internet Live Conferencing (SILC) protocol was proposed by Pekka Riikonen in 1997 (see white paper [141] and Internet-Drafts [142, 143]) for securing IRC and IM systems. The SILC Key Exchange (SKE) protocol is based on the DH key agreement and its functionality is derived from several other key exchange protocols. The SILC Connection Authentication Protocol (using a passphrase or public key) runs after a successful execution of the SKE protocol. This separation of key exchange and authentication may cause SKE to fail against a man-in-the-middle attack when the public key or certificate received during the SKE protocol is not verified. For a safe execution of SKE, users need to possess long-term public keys or certificates.

`iGo Incognito` [70] asserts itself to be an IM facility designed with cryptographically strong security. During login, a message encrypted with the server's public key is sent from a client to the server containing a user's identification parameters (e.g. user ID, password). Upon verification, a random 128-bit session key is established between a client and the server. To send a text message to a user, the sender generates a random one-time key. The sender encrypts the message with this key using AES [113]. The encrypted message is forwarded to the recipient along with the key encrypted under the recipient's public key. As the messages are relayed through the server, they are also encrypted using the shared key with the server. A client's private key is stored encrypted on the local hard drive under a user-chosen pass-phrase. Also, all user messages are digitally signed, corroborating who the sender is. However, due to the lack of available documentation, we have not attempted to independently

confirm these security properties.

2.3 Comparison of IMKE with other IM Implementations

In this section, we compare our IMKE implementation (see Chapter 7) with selected other secure IM implementations. The metrics of our comparison are:

- *Strong PAKE* (whether the authentication mechanism is a *strong* password protocol; see Table 5.3 for definition);
- *C-C Message Protection* (whether the client-to-client message authentication and encryption are supported);
- *C-S Message Protection* (whether the client-to-server message authentication and encryption are supported);
- *Mobility* (whether users can log in to the IM server with only a memorable password);
- *Repudiation* (whether users can repudiate a message);
- *No Client Change* (whether the requirement of any extra client-side software other than the IM client is avoided, or whether the IM client can be used unmodified); and
- *No Server Change* (whether the requirement of any changes to the IM server is avoided).

We compare SSL/TLS based enterprise IM clients (e.g. Reuters Messaging), AIM using client certificates, IMSecure/Trillian (self-signed certificates), Off-the-Record (OTR) Messaging (requires long-term signature keys), GPG-based Gaim-e, and SILC

with IMKE. No documentation was found for Gaim-e except its open-source implementation. We do not include iGo Incognito in our comparison for the lack of documentation. Table 2.1 summarizes the comparison of IM implementations.

	Strong PAKE	C-C Message Protection	C-S Message Protection	Mobility	Repudiation	No Client Change	No Server Change
SSL/TLS			✓	✓	✓	✓	✓
AIM Cert.		✓				✓	✓
IMSecure		✓			✓		✓
OTR		✓			✓		✓
Gaim-e		✓			✓		✓
SILC		depends*	✓		depends*		
IMKE	✓	✓	✓	✓	✓		

*SILC supports client-to-client message authentication and encryption, and repudiation, depending on client settings.

Table 2.1: Comparison of IM implementations

From the above discussion, the distinguishing features of IMKE are the following:

1. It is the only IM protocol to support strong PAKE although all IM protocols rely on passwords.
2. It secures client-client and client-server messages.
3. Although it requires changes in both the client and server software, the users do not need to maintain or *carry* any long-term public keys or certificates.
4. IMKE is not a *messaging* protocol. It does not specify anything beyond security attributes of an IM protocol. We argue that IMKE can be embedded into existing IM protocols without breaking the underlying messaging structures. This claim is supported by our implementation (Chapter 7), which offers an example of embedding IMKE with the XML-based Jabber protocol.

2.4 Password Authentication

In this section, we focus on work directly related to PAKE protocols (i.e. strong password protocols), providing a historical perspective and selected recent work on this area. More discussion on the relationship of IMKE with well-established two- and three-party protocols are provided in Section 5.1.

The idea of securing *poorly-chosen* passwords from offline guessing attacks was first introduced by Lomas et al. [92] in 1989, specifically by using a known server public key. Bellovin and Merritt [16] proposed the seminal EKE (*Encrypted Key Exchange*) protocol in 1992 (see also U.S. patent [17]) – the first strong password protocol without the requirement of a known server public key. In 1993, Gong et al. [53] refined the earlier proposal of Lomas et al. [92] and proposed a strong password protocol that does not require a known server public key. Since then, many strong password protocols have been proposed (see [75] for a comprehensive list), all using public key cryptosystems. Halevi and Krawczyk [56] provide formal arguments of the notion that public key tools are unavoidable in designing secure password protocols. The relatively short history of PAKE protocols is not untarnished, however; many flaws in novel (and sometimes "proven") proposals have been discovered.

Most of the EKE variants encrypt a dynamic public key using a password; this makes many EKE implementations vulnerable to a special form of dictionary attacks, called the *partition attack* (see Section 5.1). To prevent such attacks, Lucks [93] proposed OKE (Open Key Exchange) in 1997 which does not encrypt the public key. Protected-OKE is a variant of Lucks' solution to deal with the special case of the RSA public key system. In 2000, MacKenzie et al. [96] presented an attack on the Protected-OKE and a solution (the SNAPI protocol) to fix it. Although, as noted by Zhang [190], SNAPI is the only PAKE protocol based on RSA that has

not yet been broken, it requires a prime public exponent e larger than the RSA modulus n; thus SNAPI increases the cost of the RSA public key encryption. Zhang [190] introduced PEKEP (Password Enabled Key Exchange Protocol) to avoid this limitation of SNAPI.

Jablon [76] proposed the Simple Password Exponential Key Exchange (SPEKE) protocol, which is closely related to the Diffie-Hellman Encrypted Key Exchange (DH-EKE [16]). The main idea of SPEKE is to involve the password in computing the base used in the standard DH key agreement. Although SPEKE was "proven" to be secure [95], recently Zhang [189] showed that the fully-constrained SPEKE is susceptible to password guessing attacks when passwords are natural numbers less than or equal to a positive integer N, e.g. Personal Identification Numbers (PINs); an adversary can test multiple candidate passwords in a single impersonation attack.

AMP (Authentication and key agreement via Memorable Password) and EPA (Efficient Password-based protocol for Authenticated key exchange) protocols have been proposed by Kwon [85, 86] and Hwang et al. [65] respectively. These protocols have been designed to achieve good efficiency in addition to other strong password protocol attributes. Wan and Wang [191] showed that the password can be fully disclosed from AMP and EPA protocols by an attack based on the small factors of the order of a large group $\mathbb{Z}_p^*$; the fundamental reason for this vulnerability, as Wan and Wang stated, is that the server applies its random secret exponent on an unknown received number which may be malicious.

Many password-verifier based PAKE protocols (e.g. A-EKE [15], SRP [185]) have been designed where the server stores only an image (*verifier*) of the client password to minimize the impact when the password-verifier file is exposed. However, the disclosure of the password-verifier file allows feasible brute force searching attacks on passwords [185]. Generally, verifier-based PAKE protocols require more computation

than their plaintext variants; Boyd and Mathuria [24, p.248] note this feature as "not necessarily a significant advantage". Nevertheless, AMP [85] and EPA+ [65] have been designed to withstand dictionary attacks even when a server's password-verifier file is compromised using an additional *secure* storage device.

The IEEE P1363 Working Group has a study group entirely focused on strong password protocols. SRP, SPEKE, SNAPI, AMP etc. are among the submissions that this group will evaluate for eventual standardization (see the latest draft [67] from this group).

Chapter 3

Security Threats to IM

This chapter lists the most significant threats to public IM systems. The list is constructed from known attack forms, and IM protocol and implementation flaws that may allow future attacks. Detailed descriptions of IM exploits are available in many web resources (e.g. [61, 60, 160, 43]). One objective of compiling this list is to acquire insight to aid in designing a robust security protocol for IM systems. We discuss IM worms in greater detail in Chapter 4 due to their importance and complexity. Threats considered in our Instant Messaging Key Exchange (IMKE) protocol are summarized in Section 5.2.1.

3.1 General Threats

This section discusses threats to IM which are common in many Internet-based applications.

Insecure Connections.

Perhaps the greatest threat to IM networks lies in their open, insecure connections (see Figure 1.1 for IM communications models). IM connections are susceptible to being taken over during client-server, client-client and intra-server (especially in the distributed IM model) communications. Once authenticated during the login time, all these connections deploy little (sequence number or transaction identifier, which can be easily spoofed) or no security measures at all. Hence almost all popular IM connections lack authentication (except in the login message), confidentiality and integrity. This opens the door to many other security vulnerabilities including impersonation, denial of service (DoS), man-in-the-middle, replay, etc. For example, even if a user chooses to receive messages only from the users in his/her contact list, it is possible to inundate the user with unwanted messages; all the attacker needs to accomplish this is to capture (e.g. see [77, 14]) an open (TCP) connection with one of the user's contacts.

Denial of Service (DoS).

DoS attacks can be launched in many different ways. Some may simply crash the messaging client repeatedly. Attackers may use the client to process CPU and/or memory intensive work that will lead to an unresponsive or crashed system. DoS attacks can also be launched against the IM server (e.g. accepting new clients, providing services to online clients), and the network bandwidth.

Flooding with unwanted messages is particularly easy when users choose to receive messages from *everyone*. However, IM clients generally support user blocking. A victim can block the attacker's account ID easily; however, attackers may get through this barrier by using many compromised accounts simultaneously.

Impersonation.

Attackers may impersonate valid users in at least two different ways. If a user's password is captured, attackers can use automated scripts to impersonate the user to his/her contacts. Alternatively, attackers can seize client-to-server connections (e.g. by spoofing sequence numbers). A connection may be taken over right after a user logs in, when a user initiates a connection with a peer or when a user gets disconnected unexpectedly (e.g. by DoS attacks). The server will keep the connection(s) open for some time until the *keep-alive rate* (i.e. messages that are transferred in a certain time-interval to notify a connection's availability) is violated. Attackers can take advantage of this time out to capture the connection to the server.

As none of the popular IM services protect their connections with encryption, it is quite easy to impersonate any connection via man-in-the-middle attacks (see e.g. [60]).

DNS Spoofing to Setup Rogue IM Servers.

Malicious programs (e.g. `QHosts-1`[1]) can modify the TCP/IP settings in a victim's system to point to a different DNS server. Malicious hackers can set up an IM server and use DNS spoofing so that victims' systems connect to the rogue server instead of the legitimate one. IM clients presently have no way to verify whether they are talking to legitimate servers. Servers generally verify a client's identity by checking the user ID and password hash. This client-side-only authentication mechanism is vulnerable to (IM) man-in-the-middle attacks where a malicious server may pose as the legitimate server. Attacks such as account-related information collection, eavesdropping, and impersonation are possible if an IM server is spoofed to end-users.

[1] http://securityresponse.symantec.com/avcenter/venc/data/trojan.qhosts.html

3.2 Threats Resulting from IM Features

IM clients provide users many useful features (e.g. file transfer, application sharing) in addition to the basic text-messaging capability. In this section, we list threats originating from specific IM features.

File Transfer.

Worms can easily propagate through IM networks using the file transfer feature. Generally, users are unsuspecting when receiving a file from a known contact. Worms commonly exploit this behavior by impersonating the sender. Also, IM file transfers carrying malware penetrate firewalls more easily than email attachments. This is due to the difficulty in distinguishing IM traffic at the gateway, and IM vendors' use of proprietary protocols [60]. Like email address books, IM worms can use a user's online contact list as a propagation vector; however, unlike *offline* and slow email propagation, IM contacts provide *instant* victims for fast spreading.

YIM has an option to open a file automatically after downloading. This can help spread malcode with less user intervention. In ICQ, users can choose to automatically accept all incoming file transfer requests.

With the latest IM clients, users can set up automatic virus scanning for incoming files. However, anti-virus tools generally scan only a small subset of all possible file types. For example, a media file (e.g. an MPEG file) may contain a specially crafted data sequence that may crash a user's media player or install a backdoor program. In fact, for the Real Media [168] and JPEG [97] files, these threats are already a reality. As most anti-virus tools are not generally used to scan data files (e.g. media or image files), the widespread use of software such as Windows Media Player may become a potential source of attacks.

Due to their importance, IM worms are discussed in Chapter 4 in greater detail.

Plaintext Registry and Message Archiving.

There are security related settings in IM clients. Knowledgeable users can set these options according to their needs. IM clients save several of these settings in the Windows registry. Any technically inclined Windows user can read registry values, and users with administrative power can modify those as well.

Some security related IM settings saved in the registry are: encrypted password; user ID; whether to scan incoming files for viruses; anti-virus software path; whether permission is required to be added in someone's contact list; who may contact the user (only from the contact list or everyone); whether to share files with others; shared directory path; and whether to ask for a password when changing security related settings.

MSN Messenger even stores a user's contact list, block list and allow list in the registry in a human-readable format. Attackers can use malware to modify or collect these settings with little effort. Modifying the registry entries may help the intruder bypass some security options like file transfer permission, add contact authorization etc. By collecting user IDs and encrypted passwords, attackers can take control of user accounts. Also, the plaintext password can be easily recovered (due to the encryption process used) from the encrypted password stored in the registry using tools e.g. AIMPR [44].

IM clients generally allow message archiving. User conversations are saved in a plaintext format in a predictable system location. The revelation of these messages can be damaging (loss of message confidentiality) for corporate and home users.

Insecure Default Settings.

As is common in many software products, default security settings in IM clients are often at the low end of the client's security capability. Most IM clients allow anyone from the same IM service to contact (send text messages, files etc.) a user by default. Allowing message reception from everyone opens the door to a new vector of nuisance – *spim* – i.e. spam sent via IM systems. This option may be restricted to allow only entries from a user's contact list, because IM users do not communicate with strangers often [55]. Also, the default IM file download location in a user's machine may be exploited to run malcode e.g. as in the ICQ `scm` file vulnerability [152].

In ICQ, the default setting for the contact list authorization is "All users may add me to their Contact List and see my Online / Offline status". Clearly this is not a prudent security setting for many users. Permission for viewing a user's shared directory is set to "Only users from my Contact List" by default in ICQ. However, this does not provide meaningful security when "add contact" authorization is not required.

Sharing IM Features with Other Applications.

The MSN Messenger contact list and other features are available from applications such as the Microsoft Outlook Express email client and web-based Hotmail email service (when launched from the Internet Explorer (IE) browser). Microsoft has also published IM APIs for application developers for a custom integration of IM features with any software product. Microsoft's Live Communication Server (LCS) [104] integrates MSN, AIM and Yahoo! public IM services with Microsoft Office programs to enhance real-time collaboration in an enterprise.

Yahoo! also provides developers with programmable objects like Yahoo! Audio Conferencing and Yahoo! Webcam Upload/Viewer. AIM Express (an AIM client with minimal features) is implemented as an applet for the Java platform that runs in web browsers to support better user mobility. As IM capabilities are being integrated with many different applications, security risks are increasing for both the IM services and host applications; a security breach in an IM service can affect applications integrating IM features, and vice versa. This significantly increases attack opportunities for malcode writers (see Section 4.1).

Malicious Hyperlinks.

Links to web pages containing malicious content can be sent within normal instant messages. ICQ has an option to accept or reject messages with hyperlinks (although the default setting is to accept URL messages from all). In AIM, a user can create hyperlinks wherein the visible text is completely unrelated to the underlying web link. This can easily mislead any user receiving a hyperlink having an innocent visible text to visit a web site corresponding to a deceitful link. Also, malicious hyperlinks can vector users to phishing web sites (see e.g. [5, 119]).

URI (Universal Resource Identifier) Handlers.

YIM and AIM clients install custom URI handlers `ymsgr` and `aim` respectively. These URIs can help in writing useful scripts to be processed by applications such as Microsoft IE, Netscape Navigator, Mozilla Firefox, Microsoft Outlook, or the Windows command shell. A URI can be sent by another YIM or AIM user in an IM text-message or HTML email message. Users also can embed these URI links to their web pages. Web browsers and command shells can be used to launch AIM

or YIM to process these URIs. For example, if a YIM user executes the URI `ymsgr:addfriend?mybuddy` from IE (e.g. by clicking a hyperlink), the YIM client will be launched (if it is not already running), and the user will be prompted to add `mybuddy` to his/her contact list.

The lack of bounds checking in parameters of these protocols has allowed malicious hackers to launch various buffer overflow attacks (e.g. [175]). The program paths of YIM/AIM clients responsible for processing `ymsgr/aim` URIs are stored in the Windows registry. By changing such a registry entry to any malicious program, attackers can launch that program when these URIs are invoked. Also, scripts written using these URIs open a new front for automated attacks.

Chapter 4

IM Worms, Analysis and Countermeasures

IM worms are worms that spread in IM networks, by exploiting the features and vulnerabilities of IM clients and protocols. We use a broad definition of worms by Kienzle and Elder [81]: "A worm is malicious code (standalone or file-infecting) that propagates over a network, with or without human assistance".

It is well-known that IM worms are on the rise. The first IM worm made news in August 2001 (see e.g. [177]). Since then, IM networks have slowly become potential breeding grounds for spreading worms. As of March 2005, they are being reported so often (see e.g. [167]) that major public IM vendors (including Microsoft, AOL and Yahoo!) have formed an IM Threat Center [69] to track the latest IM and P2P worms and vulnerabilities. A malware evolution analysis [54] for the first quarter of 2005 reports that email worms are on the decline due to improved anti-virus products; in contrast, IM worms are becoming common.

Due to the prevalence of scanning- and email-worms, security researchers have employed large efforts to understand and restrict the propagation of those worms

[156, 151, 136]. Although the war is far from being won, improved defensive techniques are making it increasingly difficult for worm writers. On the other hand, there appears to be very little published research detailing efforts to understand and contain the spread of IM worms.

In this chapter, we provide an overview of selected IM worms and vulnerabilities, and summarize major replication mechanisms for IM worms. Distinguishing features that make IM networks susceptible to fast-propagating worms are also discussed. We briefly discuss theoretical characteristics of IM networks to understand worms' behavior in such networks. A critical review of existing mechanisms to address IM worms (e.g. virus-throttling for IM [184]) is provided. We propose two new techniques to restrict the spread of IM worms. We also provide the results of a three and a half years user study of IM text messaging and file transfer frequency in a moderate-size public IM network – the largest such study to date – which is of independent interest, but also supports in part the usability claim of our proposed new techniques. Note that, the techniques we outline here are aimed to restrict IM worms' propagation; we propose our secure IM protocol (IMKE) in Chapter 5 to reduce other IM threats. The results in this chapter largely appeared in WORM 2005 [100].

4.1 Analysis of Selected IM Worms and Vulnerabilities

In this section, we briefly analyze a few IM worms.[1] We begin by mentioning a few IM worms and noting their distinguished characteristics. Then we list a few IM clients' vulnerabilities (exploitable IM features or implementation bugs) which could be used

[1]Details are available on many security web sites (e.g. [54, 160, 171]), or via search engines (e.g. Google).

to write more damaging and faster spreading IM worms. From our discussion, we identify the most common propagation mechanisms of IM worms.

Examples of IM Worms.

1. The `W32.Choke` (June 2001) worm [164] hooks to MSN Messenger, and when a user initiates an IM conversation with an infected host, the worm sends a text message along with an invitation to download a file (which turns out to be the worm-file) from the infected host.
2. The `W95.SoFunny` (July 2001) worm [166] spreads as a file attachment using AIM. It steals AIM login information, and emails the user ID and password to the worm author. It runs as a *service process* in Windows systems to hide itself from the Windows Task Manager.
3. The `JS_Menger` (Feb. 2002) worm [170] appears as a message in MSN Messenger with the URL of a web site hosting the worm code. It exploits a vulnerability in the Microsoft Internet Explorer (IE) browser.
4. Different variants of the `Bropia/Kelvir` (Jan. – Feb. 2005) worms attempt to spread to all online contacts as a file using MSN Messenger. `Bropia-M` [163] disables several system processes in a compromised host, including Windows System Configuration Utility, Registry Editor and Task Manager. `Bropia-F` [173] disables debugging tools in an infected system, and installs a variant of the `Agobot/Spybot` worm [172], opening a backdoor on infected systems. The Agobot worm propagates by exploiting many known vulnerabilities e.g. SQL server buffer overflow, IIS (Internet Information Services) or WebDAV (Web-based Distributed Authoring and Versioning) vulnerabilities, and can be used to log keystrokes and relay spam.

5. Several `Serflog` variants (Mar. 2005, e.g. `Serflog.A` [165]) spread via MSN Messenger (as a URL) or P2P file-sharing networks (e.g. `eMule`[2]). It terminates anti-virus and other well-known system and security processes, and modifies the system's HOSTS file to block access to well-known security web sites.

Most IM worms install backdoor programs to further exploit infected hosts. However, to date such worms generally have not carried any damaging payload other than the replication engine. Social engineering techniques are typically used to convince users to accept a worm file transfer or visit a malicious URL link.

Examples of Client Vulnerabilities.

Numerous vulnerabilities have been uncovered in popular IM clients. These are drawing the attention of IM service providers and security communities who recognize that worm writers could easily exploit these vulnerabilities to write more damaging IM worms.

As an example, buffer overflows were discovered (and subsequently fixed) in the past in AIM and YIM clients (see e.g. [60, 161, 120]). In a recent (May 2005) buffer overflow attack on YIM, attackers could delete files from an infected host [122]. The ICQ sound scheme file location vulnerability [152] could be exploited to place malicious content on a user's machine. This could allow attackers to run a malicious script in the compromised machine exploiting a known IE vulnerability. The `Bizex` worm [162] exploited this vulnerability to spread in the ICQ network.

While most known client vulnerabilities require user-actions to exploit a host, the recent (Feb. 2005) MSN Messenger PNG (Portable Network Graphics) vulnerability (see e.g. [106]) could have been exploited to spread a worm without any additional

[2] http://sourceforge.net/projects/emule/

user-interaction. MSN Messenger uses the PNG format to enable the "Display Picture" feature (which shows a small picture of a contact in a chat window). The picture data is sent to a user whenever the user wants to send text messages to a contact even though the recipient's image may not be displayed in the sender's chat window (depending on the sender's IM client settings). Therefore, disabling this feature in an MSN Messenger client could not prevent the exploitation. Microsoft fixed this vulnerability and forced users to update the IM client; users who did not upgrade were unable to log in to the MSN Messenger service.

Similar to the PNG vulnerability, the MSN Messenger GIF image processing vulnerability (April 2005) could lead to remote code execution [107]. This vulnerability can be exploited by sending a maliciously crafted *emoticon* (i.e. emotional icons in text messages) or display picture to an online MSN Messenger client. Microsoft subsequently updated the IM client software to fix this bug.

IM Worm Replication Mechanisms.

From the above discussion, the following are evidently among the most popular mechanisms for propagation of IM worms.

1. Malicious file transfer (requires user-action).
2. Malicious URL in a text message (requires user-action).
3. Exploitation of implementation bugs in IM clients (may or may not require user-action).
4. Exploitation of vulnerabilities in operating systems or commonly used software applications (may or may not require user-action).

It follows that mechanisms for limiting the spread of IM worms (see Section 4.4) should take into account these vectors.

4.2 Distinguishing Features of IM Networks

We now list four major reasons that IM networks are particularly vulnerable to a worm attack.

A) Popular and Connected.

Public IM networks connect millions of users. According to Microsoft, MSN Messenger has 130 million customers as of July 2004 [105]. As more users continue to move to high-speed Internet connections, more remain online for extended periods of time [22]. Therefore, the set of online IM users forms a larger and larger network as time passes with one consequence being that the number of potential victims possibly outnumbers any known worm-outbreaks to date.

B) Instant Hit-List.

IM contact lists enable users to track the *presence* status of their contacts. To a worm, an online contact list provides an instant *hit-list* [156] – a list of users vulnerable (with a very high probability) to the worm from an infected host. Most IM users are connected through a homogeneous platform (i.e. the same operating system and IM client), which increases the success ratio of the hit-list. For example, online users in the contact list of an MSN Messenger client are mostly connected in the MSN IM service using the default MSN Messenger client on Windows. Users can access the MSN IM service from Mac (using the MSN Messenger for Mac [103]) or Linux (using e.g. `aMSN` [124]) machines. In present reality, however, the number of users accessing the MSN IM service using clients other than the MSN Messenger in Windows is negligible.

To achieve a very rapid initial rate of infection (i.e. *getting off the ground*), a hit-list of vulnerable machines is required [156]. A worm writer could build a hit-list by scanning the whole Internet for vulnerable hosts before releasing a worm in the wild. By the time the worm hits the Internet, however, the hit-list may be stale, at least in part; some machines from the hit-list might be turned off, or some might be *patched* against the targeted vulnerability. In contrast, an IM contact list offers a worm writer a reliable hit-list "for free" (i.e. without spending any time to build it).

A simulation performed by Symantec [61] (with relatively conservative assumptions) shows that 500,000 machines can be infected with an IM worm in approximately 30 seconds. At least in theory, IM worms could out-perform email- and scanning-worms, which are typically limited by their ability to find vulnerable hosts across the entire Internet. IM worms would most likely be spread by using TCP connections (as IM connections are mostly TCP), and would thereby be limited by the network latency. Nevertheless, due to the inherent availability of a highly accurate hit-list, IM worms can spread at a high rate.

C) Reduced User-Interaction Time.

Despite numerous security measures, email remains one of today's most effective worm propagation vectors [81]. This is despite the fact that an email worm, sent as an attachment, is not assured of reaching an intended destination address (e.g. the target email address may be invalid) or of immediate user-action (e.g. the target user may check email only occasionally). In contrast, as noted above, an IM worm gets a largely accurate instant hit-list from an infected user's online contact list. A file transfer request or a URL sent to an online contact results, with high probability, in an instant user-action (despite some online IM users being away from their machine). Thus, reduced user-interaction time for propagation of worms is expected in IM networks.

D) Increasing IM Integration in Popular Applications.

IM features are being integrated with many popular applications to meet the desire of IM users to share ideas and information instantly. For example, the MSN Messenger contact list is accessible from the Microsoft Outlook Express email client; Microsoft's Live Communication Server (LCS) [104] integrates MSN, AIM and YIM public IM services with Microsoft Office programs to enhance real-time collaboration in an enterprise. Greater integration increases usability, although the chance of IM-integrated applications being used by many users unknowingly or unwillingly increases as well. A security vulnerability in an IM protocol or client can exploit applications integrating IM features, and vice versa.

Although IM is just another Internet based application, the distinguishing features as listed above make IM networks particularly vulnerable to a fast-spreading worm.

4.3 Topology of the Network formed by IM Contacts

We now provide an overview of the network of IM contacts. Understanding the topology of such complex networks is necessary to obtain a better understanding of the rapidity of the spread of a worm in an IM network. This may, as well, help in devising novel ways to prevent or contain such an epidemic.

Paul Erdös and Alfréd Rényi introduced the ER (Erdös-Rényi) *Random Graph Model* in 1959 to analyze complex networks which are evident in communications and life sciences. ER networks (also known as *random networks*) are formed by nodes with randomly placed links, with most nodes having an approximately equal number of links. In the late 1990s, however, empirical network data (e.g. from the

World Wide Web [2]) suggested that a common property of many complex real-world networks is that node linkages follow the power-law distribution[3] instead of the Poisson distribution of random networks. Barabási and Bonabeau [7] coined the term *scale free* (SF) to model the topology of such complex networks. The SF model originates (see [6]) from the following two mechanisms: networks grow continuously by the addition of new nodes, and new nodes attach *preferentially* to more highly connected nodes. This model accommodates the existence of *hubs* or *clusters* (i.e. a few nodes with a very high number of links, while most nodes have a few connections only) in real networks. SF architectures (e.g. WWW, Internet routers) are found to be highly resistant to accidental (i.e. random) failures, though very vulnerable to targeted attacks. Also, all viruses, even those that are weakly contagious, will spread and persist in a SF system. Consequently, once introduced, computer viruses are very difficult to completely remove from the Internet (see [134, 7]).

A general observation in human social structures is that some friends of a given person are typically friends themselves. Contacts in a user's IM contact list also exhibit this feature; i.e. if user A has users B, C, D in her contact list, B will have C and/or D in his contact list with a high probability. Smith [155] analyzed the topology of the network formed by IM contacts, and found the network to follow the SF model. He studied a Jabber IM network of 50,158 nodes (young adult users) with nearly 500,000 links (contacts). The diameter of the network was only 4.35; i.e. any two users in the network studied were separated by only 4 to 5 users on average.

While IM networks may exhibit the SF model's properties in reality, the following aspect of IM worms may complicate such a model. From any infected user, a worm's spread may not be limited by the number of online contacts of that user; an IM worm may successfully guess IM contacts, at least for large public IM services (see

[3] i.e. the probability that any node of the network is connected to k other nodes is proportional to $k^{-\gamma}$, where γ ranges between 2 and 3.

shortcomings of virus throttling in Section 4.4.1 for further discussion). In fact, each node can potentially become a hub, thereby dramatically increasing the number of hubs in the SF model. This may result in the failure of virus halting measures designed for SF networks (e.g. [38]) that aim to restore a *finite* epidemic threshold by patching most of the hubs in an infected network.

4.4 Measures for Limiting IM Worm Epidemics

In this section we analyze mechanisms found in the existing literature that are proposed to limit IM worm-outbreaks. We also outline techniques complementary to IMKE, our proposed secure IM protocol, to effectively restrict worms in IM networks.

4.4.1 Existing Techniques and Remarks Thereon

An overview of existing techniques, as well as their shortcomings, is given below.

A) Temporary Server Shutdown.

All IM connections go through or are initiated by an IM server. A typical (but effective) solution that takes advantage of this centralized structure to stop an IM worm-outbreak is the following [61]: shutdown the IM server, manually analyze the worm and build a client patch; then force update when users attempt to login as the IM server comes alive again. For example, in April 2005, Reuters Messaging was shut down for several hours to stop an IM worm spread (e.g. [121]). Obviously, this is not a user-friendly solution. Moreover, as it has already been observed, IM worms may enable the propagation of other worms exploiting known (and generally ubiquitous) vulnerabilities in infected systems, and spread via traditional mechanisms

such as email attachment. Therefore, shutting down the IM server may not effectively contain an IM worm epidemic.

B) Temporarily Disabling the Most-Connected Users.

Exploiting the scale-free nature of IM contacts, Smith [155] proposed the following measure to contain a worm outbreak: disconnect the most-connected users (determined by the size of contact lists), which may effectively increase the network's diameter, and thereby slow down the spread and allow time to build and apply a patch. It has been shown by Smith, however, that disabling the top 10% connected users in the IM network still would leave 90% of the remaining network connected although the network's diameter has increased almost twofold. Note that Smith's study derives users' connectivity from the total number of contacts, not from the number of online contacts (which may vary significantly); of course, an IM server can easily also track the most-connected online users.

C) Virus Throttling for IM.

To limit the propagation of fast worms, Williamson [183] introduced a general throttling mechanism based on the observation that worm-spreading network traffic is significantly different than normal traffic on many Internet protocols (e.g. TCP/IP, email). Generally, IM users interact with a slowly varying subset of contacts while IM worms send messages to all the online contacts in a user's contact list. The idea behind virus throttling for IM, as proposed by Williamson and Parry [184], is to limit the rate at which users can interact with their contacts in a way that impedes a worm-spread (even for unknown, *zero-day* worms) without noticeable impacts on IM users.

The Throttling Mechanism. In essence, the throttling mechanism for IM is as follows. A short recent history list or *working set* of contacts for each user is maintained. This is the list of contacts that a user has *recently* interacted with. The mechanism attempts to ensure no more than one message is sent to a new contact (a user from the contact list who is not in the working set) per time period; see Figure 4.1. Every time a user attempts to send a message to a contact, the recipient's user ID ('h' in Figure 4.1) is compared with the working set. If it is in the set, the message is sent immediately; otherwise, the message is placed on a delay queue for sending later. The delay queue and working set are updated regularly. If a client's delay queue length exceeds a predefined threshold, all messages from that client are blocked and the user is asked to confirm the legitimacy of the queued messages, assuming an attempted worm-spread. It is suggested that the mechanism be implemented at the IM server, because the IM server processes or initiates all user-messages, and users would not be able to bypass the server.

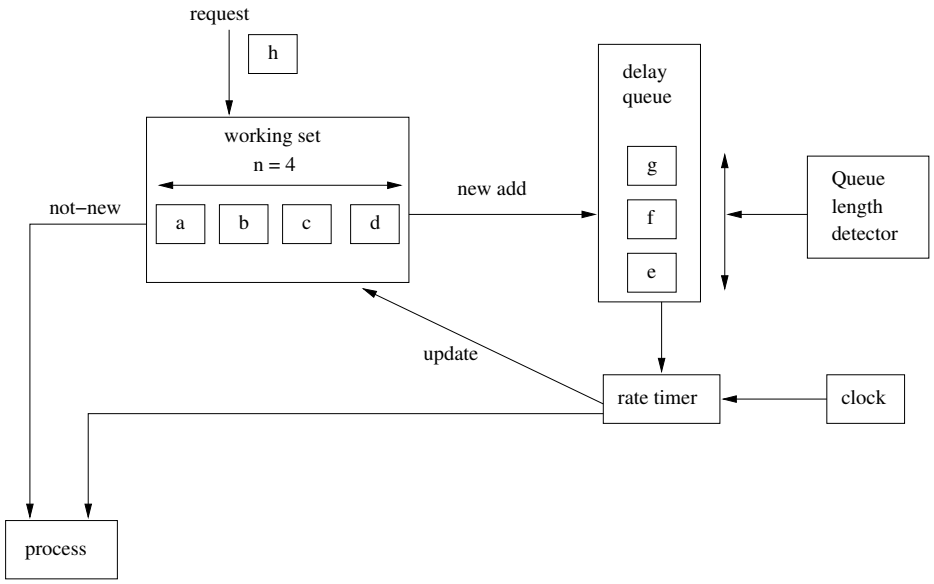

Figure 4.1: Throttle algorithm for IM [184]

Data Analysis. Williamson and Parry collected IM usage data from the Jabber

server deployed at Hewlett-Packard. Contact lists of 710 corporate users and 39,740 messages from 223 senders in a 72 day period were analyzed. Setting a working set size to five resulted in less than 10% of messages being placed in the delay queue for most of the users. Although setting the allowed rate at around one new contact per minute appears more user-friendly, to restrict a worm-spread effectively, Williamson and Parry suggested the allowed rate to be one new contact per day. In either case, some messages will escape before the throttle mechanism is applied. Also from the collected data, the network formed by IM contacts was shown to be scale-free. The *hubs* of this network (i.e. users with large contact lists) appear to release the fewest messages before being blocked when the throttle is applied. This property apparently provides evidence that the throttling mechanism may effectively restrict IM worms.

Shortcomings. Throttling for IM may help stop IM worms to some extent, and appears to be one of the few technical mechanisms proposed to date. Thus it provides an important benchmark for comparison. However, we note the following limitations.

1. Allowing only one new contact/day may be *too restrictive* for IM users (home or enterprise). If users need to confirm the legitimacy of messages *often*, the throttling mechanism may find little acceptance in IM communities.
2. Delaying a message temporarily (e.g. a minute) is not what IM users expect in an IM environment; the whole point of using IM is being *instant*.
3. It is assumed in throttling that IM worms' propagation vector will be limited to users' online contact lists, because IM worms can guess new contacts with only a very low probability. For example, in AIM a user ID consists of (case-insensitive) alphabetic characters and digits, and the maximum allowed length is 16; i.e. AIM supports approximately 36^{16} different user IDs, so randomly guessing a user ID correctly would appear to have a very low probability. In reality, however, only a small (typically predictable) fraction of the user ID

space is used. For example, any dictionary word or common name guessed as a user ID has a significantly higher than random chance of being correct. The large number of users in public IM networks makes such guessing of correct user IDs easier. Also, a large number of online user IDs can easily be obtained, for example, through the chat-room feature (e.g. YIM chat-rooms).

4. A user must confirm the legitimacy of messages when that user's delay queue threshold is exceeded. A worm may be able to do that on behalf of the user, and thereby compromise the throttling mechanism.
5. The size of IM usage data analyzed by Williamson and Parry is quite small, and appears to be unrepresentative of active IM usage. For example, their study's average of less than 2.5 messages per user per day appears to largely fall short of the number of messages in real IM practice (our study in Section 4.4.3 shows that in a mid-size public IM network, on average a user sends 334 text-messages per day). Also, the network of IM contacts formed by only 710 users does not likely represent the characteristics of a public IM network with millions of users.
6. Throttling does not deal with group-chat or chat-room situations, where a user sends a message to multiple recipients concurrently.
7. Implementing the throttling mechanism, as suggested, requires per-online-user state memory at the server (e.g. working set, delay queue).

We have discussed several limitations of available techniques designed to limit IM worm-outbreaks. We briefly propose containment strategies in the next subsection to avoid most of these shortcomings.

4.4.2 New Proposals

In this section, we briefly suggest two mechanisms (complementary to IMKE) to further reduce the propagation of IM worms, and to do so in a more user friendly

manner. While these mechanisms may appear to be straight-forward, we record them nonetheless, as an early contribution to the scant literature in this research area.

Our discussion in Section 4.1 suggests that file transfer and URLs in text messages are two major vectors for malware propagation in IM networks. It is apparent that, if we can restrict automatically generated file transfer requests and text messages with malicious URLs, the propagation of IM worms can be significantly delayed. In essence, our new proposals aim to control these two vectors. We assume that all IM text messages and file transfer requests (not necessarily the file data) are relayed through the IM server – which is generally true in current public IM connections (recall Section 1.2).

In Section 4.4.1, we listed the limitations of the virus-throttling mechanism for IM. Nevertheless, we believe that such a mechanism can do little while working on top of an insecure IM protocol. Similarly, a secure IM protocol alone is unlikely to eliminate the possibility of IM worms because IM worms may take advantage of implementation bugs of a secure IM client. A two-step method – a throttling mechanism as well as a secure IM protocol, e.g. IMKE – appears more effective to us against IM worms. Here we propose the following techniques which are derived from the throttling mechanism or similar to it in spirit, and which we recommend be deployed with IMKE or a similarly-featured protocol.

1. Only use a throttling mechanism to limit file transfers and messages containing URLs. We believe this will be as effective as the proposal of Williamson and Parry [184] in most cases with less impact on users, due to the general observation that IM users send file transfer requests (e.g. see Section 4.4.3) or text messages with URLs only occasionally. In other words, we propose that the throttling mechanism be applied only to file transfer requests and URL messages, instead of on all new connection requests. Now normal text messages

(the most frequently used feature of IM) will not be delayed, and there is no threshold at which users must confirm the legitimacy of text messages. Instead, URL messages and file transfer requests may be delayed, and in the case of file transfer, the usability impact is further minimized due to the fact that a file transfer is generally not *instant* (even for a small size file). When sending a file transfer request or URL message to multiple users at a time (e.g. in a group-chat), the sender needs to confirm the authenticity of the request (to the server or a selected recipient) only once.

2. Challenge senders of a file transfer request or URL message with a (e.g. visual) CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) [169] generated by the IM server or the recipient's client. This will not affect usability much, if as noted above, these features are used infrequently. CAPTCHA tests are computer generated, easy for humans to solve but difficult for machines, and the probability of guessing the answer correctly is low. Answering a CAPTCHA correctly increases the chance of the sender being a *human* user, rather than an IM worm trying to automatically propagate using the sender's user ID. IM users are no stranger to these tests as while registering for an IM account, all major public IM vendors challenge users with CAPTCHA tests to stop automated account acquisition by software bots (e.g. by spammers).

To expand the latter technique, we note the following. Some users may send more files than others. Challenging such users for each file transfer with a CAPTCHA is less user-friendly than desired, and may be avoided by using *secure cookies* in a manner similar to Pinkas and Sander [137]. Assume that user A wants to send a file to user B, and both are logged in to the same IM server S. When A sends her file transfer request to B through S, S challenges A using a CAPTCHA. If A successfully

responds to the challenge, then S forwards A's request to send the file to B; S also sends a secure cookie (i.e. data encrypted symmetrically under a key known only to the server) to A, and A stores that cookie to the local system. The cookie will expire after a certain interval, e.g. 12 hours. If A sends more files to B, or any other user before the expiry of the cookie, S will not challenge B. If A wants to send a file to multiple recipients at a time (e.g. while in a group chat), A will be challenged by S only once.

To reduce the load of generating a large number of CAPTCHAs by the IM server, the recipient's IM client can be used instead to generate the challenge CAPTCHA sent to the originator, as user machines (e.g. personal computers) are generally under-utilized.

4.4.3 User Study on Per-User Frequency of IM Text Messaging and File Transfer

Here we summarize our findings for text messaging and file transfer statistics in an IM service, from a new IM user study which we believe should be of independent interest. Usability issues related to our proposals (of Section 4.4.2) are also examined. Although it is generally believed that the number of file transfer (FT) requests in an IM network is far less than the number of text-messages (TM), we found no real data supporting this belief in the literature. Therefore we carried out an independent study to quantify the file transfer usage in a reasonable-size IM network.

We used (with permission) data collected from the Eyeball Chat [45] service for a period of three and a half years (Sept. 2001 to April 2005). The data used contained only *collective* statistics that do not violate privacy of Eyeball's users. While the primary focus of this service is to enable *live* video conversation over the Internet,

it nevertheless supports text messaging and file transfer. In total, 17.84 million FT requests and 3.2 billion TMs were tracked. The average number of FT, TM and online users over 15-minute periods is given in Table 4.1. The number of online users in the table is the average of the maximum and minimum online users in the 15-minute intervals. Table 4.2 shows that the ratio of FT to IM is quite low (i.e. one FT per 181 TM, on average). Also, per 24-hour period, an online user in this dataset sent 334 text-messages, and 1.84 file transfers on average.

Feature	*Avg. Number*
File Transfer (FT)	143
Text Message (TM)	25953
Online Users	7459

Table 4.1: Average file transfer, text messages, and online users over 15-minute intervals

Ratio	*Value*
FT/TM	0.0055
FT/user/day	1.84
TM/user/day	334.03

Table 4.2: Comparison of file transfer (FT) and text message (TM) usage

From this it is apparent that IM users (at least in this dataset) infrequently transfer files, although it is a useful feature of IM. Hence throttling file transfers or challenging the initiator (sender) of a file transfer with a CAPTCHA would appear to be practical. As one of our proposals involves the use of a CAPTCHA for improved defense against IM worms, the security of such tests must be taken into account, as another ongoing arms race is that between the designers, and attackers, of CAPTCHAs (e.g. [109]).

Our proposed mechanisms may be applied to limit automatic URL propagation as well. However, the usability impacts for CAPTCHA-protected URL messages are not known, because of the unavailability of such usage data.

Chapter 5

IMKE: Instant Messaging Key Exchange

The shortcomings and security risks of public and business IM protocols, as discussed in the previous chapters, highlight the need of a secure IM protocol. The techniques outlined in Chapter 4 to restrict the propagation of IM worms improve IM security only in part. To complement those techniques, and to reduce other IM threats (recall Chapter 3), a secure IM protocol is essential. In this chapter, we present a new protocol called Instant Messaging Key Exchange (IMKE) – which we propose as an efficient protocol for strong authentication and "secure" communications (see Table 5.3 for definitions) in IM systems. IMKE may be used as a Password Authentication and Key Exchange (PAKE) protocol, and in server-mediated P2P communications. We explain what motivates our protocol as well as the differences between IMKE and PAKE protocols. Protocol goals and the threats IMKE is designed to counter are also discussed. A security analysis of IMKE is provided in Chapter 6.

5.1 Motivations for IMKE

In this section, we discuss the motivations for IMKE, and similarities and differences of IMKE with existing two- and three-party PAKE protocols.

IM authentication is password-based, and there are many PAKE protocols for strong authentication. One possibility is to use such a well-known protocol; the advantage is that well-known protocols have generally been peer-reviewed and scrutinized. Some are also supported by security "proofs", although such a proof does not necessarily guarantee security; see Koblitz and Menezes [83] for an interesting analysis of "provable security".

A *pluggable* security protocol – implemented in a third-party client software add-on module that does not require any changes in popular IM clients and servers – could easily be deployed at the client-end. Even though security needs are increasing, it would be difficult to convince major IM vendors to replace their IM clients and servers. Therefore many initiatives (see Section 2.2.2) have been taken to make IM secure using pluggable security protocols. Limitations of these solutions are discussed in previous chapters. A new, *light weight* protocol which can easily be embedded into existing IM protocols (by the IM service providers, changing both the IM client and server) seems practical to achieve security without limiting usability and requiring a large implementation effort. We propose IMKE to achieve such objectives and avoid limitations (e.g. the requirement of long term user public keys, insecure client-server connections) of pluggable solutions.

In summary, the design of IMKE has been inspired by following facts:

1. Existing IM security solutions are inadequate to address IM threats;
2. A pluggable IM security protocol can provide only limited security;
3. Well-known PAKE protocols do not directly fit into the IM communications model (see below); and

4. A *minimal* security protocol, which can conveniently be embedded into popular IM protocols without breaking the underlying messaging structures, is essential for a greater integration.

5.1.1 Relationship of IMKE to Two- and Three-Party Protocols

IM is essentially a three-party system – the IM server's main role is to enable trusted communications between users. In traditional models, the requirement of a third-party is considered a disadvantage and even when a third-party is present, it is often considered as a *disinterested* party [12]. In contrast, many of the present Internet communications (e.g. web browsing, email) involve a server – a trusted but not so disinterested party. Especially in an IM environment, the IM server plays an active role in users' communications. Therefore we take advantage of the presence of an active IM server in IMKE, e.g. by using the server as a trusted public key distribution center for clients.

Another major difference of IMKE with other three-party systems (e.g. GLNS [92, 53], 3PKD [12], key translation centers [101, p.547]) is that, although the IM server in IMKE helps establish a secure session between two clients, the server does not know the session key shared between the communicating parties. This is a desirable property for consumer IM networks; users may want their conversations to be inaccessible to the IM server even though they must trust the server for login, sharing user-profile, forwarding presence information etc. Also, in most of these three-party protocols, a user must send some protocol messages directly to the other party. Because establishing direct P2P connections may not be possible in many network settings (e.g. due to firewalls), we cannot use these three-party protocols

without modifications.

In a typical three-party case, two users start a session (authenticating themselves to the trusted server, and each receiving a server-generated session key) only when they need to communicate. The IM scenario is a bit different in the following way: users authenticate themselves only when they login to the IM server; then users initiate sessions with other online users whenever they wish to. Logging in to the IM server does not necessarily follow IM sessions (e.g. text-messaging, file transfer).

The two-party PAKE protocols that use a known server public key (e.g. Halevi-Krawczyk [56], Kwon-Song [87]) have similarities with IMKE. These protocols, as well as two-party password-only authentication and key exchange protocols (e.g. SRP [185, 186], AuthA [13], PAK [26]) may be transformed into a three-party protocol in the following way [24, p.267]: run two two-party protocols between the server and each of the communicating principals; then use the established secure channels to distribute session keys or any other communication primitives e.g. public keys among users, thereby providing the communicating users a secure channel. However, we are interested in the best possible practical protocols, and thus seek a more efficient protocol, as these solutions may require up to three extra messages per protocol run – one for sending a client's public key to the server and two for verifying the public key.

5.1.2 Relationship of IMKE to EKE and Similar Protocols

IMKE has similarities in setup and goals with many PAKE protocols related to or inspired by EKE [16], especially with those that use a known server public key. Boyd and Mathuria [24, p.276–281] provide a concise overview of password-based protocols using a server public key. We now highlight some of these similarities and differences of

IMKE with other well-known protocols. In comparing IMKE with EKE-like protocols, we only consider the PAKE phase of IMKE (see Section 5.3).

The basic idea of the EKE solution is the following. Alice (A) generates a random public/private key pair and sends the public key to Sam (S) encrypted using a shared password P (or a function of P) as the key in a symmetric cryptosystem. Using P, S decrypts the received ciphertext and obtains A's public key. Then S encrypts a random secret R using A's public key and then again using P, and sends the resulting ciphertext to A. A recovers R, using both her own private key and the shared password. A session key is then derived from R using standard techniques.

An effective way to break many implementations of EKE is the *partition attack* [16, 25, 190] (also known as the e-residue attack): a special class of the dictionary attack, where an adversary tries to partition the password-space into feasible and infeasible sets by using information gathered (passively, from the wire) from a protocol run; the correct password may be recovered from the feasible set of passwords in logarithmic time after observing a limited number of valid protocol runs. If the plaintext has any redundancy, or lies in a constrained space, then a partition attack on the password (used as the key to encrypt the plaintext) can be mounted by collecting the resulting ciphertext from several valid protocol runs. As a public key usually contains distinct redundancy, many implementations of EKE (mainly RSA-EKE, although other EKE implementations are not fully immune) are susceptible to this attack (e.g. [135, 25]). As noted by Patel [135], attacks against RSA-EKE are *feasible* – for a dictionary of size one million, only 18 successful EKE sessions (on average) leak enough information to crack a password. Patel also stated that RSA-EKE can be broken irrespective of RSA parameters, but DH-EKE and ElGamal-EKE can only be attacked if related security parameters are not verified (e.g. g is not a generator). Also, in EKE implementations some parameters must be sent in the clear (depending on the public key cryptosystem

being used). For example, in RSA-EKE, the modulus n must be sent in the clear to minimize threats from partition attacks. IMKE implementations do not require any such restrictions.

In IMKE, each message ensures the identity of the sender – a client or a server does not reply blindly. In contrast, for example, in EKE (as described above), S decrypts the first message from A using P and obtains A's public key. As noted by the EKE authors [16], it is not always practical to test the validity of a public key. Hence, in EKE, after the first message, S has no idea who is at the other end of the communication line. This type of blind reply sometimes enables an attacker to conveniently extract artifacts related to a user's password from the server.

In EKE and its variants, the user-chosen (often weak) password is used as an encryption key. As noted by Halevi and Krawczyk [56], one major difficulty in designing PAKE protocols is the assumption that the underlying crytosystems remain secure even when we use low-entropy passwords as cryptographic keys. Gong et al. [53] discuss how padding and verifiable plaintext issues can open a protocol to a variety of attacks when passwords are used as cryptographic keys. IMKE avoids this problem by not using the password as a cryptographic key.

The main idea behind our IMKE protocol is the following. We seek to break the number theoretic relationships between a public key and a password, mainly to avoid known and unknown security issues in this regard. IMKE uses a known server public key to encrypt a random (session) key (e.g. 128 bits) and uses that key to encrypt the (weak) user-password and the user's dynamic public key. This enables IMKE to avoid the partition attack because to discover redundancy in a public key, now the attacker must search for the correct session key which is generated from a large key space instead of the relatively small password space. Also, as clients' public keys are generated dynamically for every login attempt, users do not need to maintain any

long-term public keys.

5.2 Setup for IMKE

In this section, we discuss threats considered in IMKE. We list the notation and terminology used, the end user goals, and long- and short-term secrets for IMKE.

5.2.1 Threats Considered in IMKE

Table 5.1 summarizes the most significant IM threats and whether a threat is addressed by IMKE. We defer a more concrete discussion of the IM threat model to Section 6.1.1. Details of these threats are discussed in Chapter 3 and 4.

Threats	*Addressed by IMKE*
Insecure connection	✓
Limiting denial of service (DoS)	✓
Replay of messages	✓
Impersonation of IM users	✓[a]
Limiting the propagation of IM worms	Partial [b]
DNS spoofing to setup rogue IM servers	✓
Insecure default settings on IM clients	✗
Sharing IM features with other applications	✗
URI handlers (`aim`, `ymsgr`)	✗
Plaintext registry and archived messages	✗

[a] Assuming no theft of users' passwords, including e.g. through the use of keyloggers.
[b] For worms which propagate through automated file transfers and URLs within messages. IMKE helps techniques outlined in Section 4.4.2 to be more effective by securing IM connections.

Table 5.1: Threats to IM and those addressed by IMKE

The network messages in an IM system may be categorized as *control* (e.g. login) and *data* (e.g. text, status) messages. IM connections generally involve a client and a server, or two clients. Most IM threats arise from these connections being easily compromised. Our goal is to provide confidentiality, authentication and integrity protection for all IM data messages. The security related goal of availability is beyond the scope of our work. DoS attacks against IM clients or the server are not fully addressed by IMKE. However, the server and clients can attempt to limit the extent of these attacks by dropping any connection that fails to meet authentication and integrity goals. An attacker may replay captured messages (from an ongoing session or older sessions) to clients or the IM server. Replay attacks are also detected in IMKE.

An attacker can capture a user's password using a keylogger, i.e. a program or hardware device specialized in (secretly) recording keystrokes. Very few, if any, security guarantees can be provided in environments susceptible to keyloggers. However, threats from keyloggers are not insignificant (e.g. see [158, 117]). Also, attackers may collect passwords using social engineering techniques (e.g. [63, 131]), or malicious software that scans memory (or the Windows registry). Impersonation using a stolen/compromised password cannot generally be prevented in password-only systems as a password is the only piece of secret shared between a user and the IM server. However, impersonation attacks based on compromised connections can be prevented by securing IM connections.

Automated file transfer requests and hyperlinks initiated by IM worms can be limited by applying techniques outlined in Section 4.4.2. Automated propagation of hyperlinks for other malicious purposes e.g. phishing is also restricted by those techniques. IMKE complements those techniques by securing IM connections. However,

malicious files or hyperlinks manually sent from other users (knowingly or unknowingly) are not addressed by IMKE or the techniques limiting IM worm propagation.

An attacker may spoof DNS entries in a user machine (the local DNS cache) to redirect all communications to a rogue IM server. IMKE prevents this attack from being successful by authenticating the IM server to the users by using a shared secret (password), and verifying the known server public key (online).

Default settings can be dangerous, if not set appropriately. Sharing IM features with other applications increases user-interactivity; nevertheless, it introduces significant security risks. Custom URI handlers may open up new methods of scriptable attacks on IM systems. Plaintext registry values and archived messages may expose security sensitive information to malicious programs. IMKE provides no protocol level protection against these attacks.

5.2.2 Notation, Goals and Secrets

In this section, we specify the end-user goals (Table 5.2), and the required secrets as well as the notation and terminology (Table 5.3) used in IMKE. Fulfilling the end-user goals corresponds to the threats we consider in Table 5.1. We outline how IMKE achieves goal G1 through G4, and G8 in Section 6.1. The discussion of IMKE protocol messages in Section 5.3 shows how the protocol establishes goal G5 and helps against replay and DoS attacks (goal G6 and G7).

A password (user-chosen, generally assumed to be weak) is shared between an IM server and a user. This is the only long-term secret for users and they choose their initial passwords during the IM account setup (using a out-of-band method). A user may change the password whenever he/she wishes to do so. The server stores original passwords. The other long-term secret is the IM server's private key (for decryption). A server public key generally remains valid for a long time (a year or more), and a

G1. Assurance of server's and clients' identities to the communicating parties without exposing clients' passwords to offline dictionary attacks.

G2. Secure communications* between a client and the IM server.

G3. Secure communications for messages flowing directly between clients (cf. G5).

G4. Forward secrecy and repudiation.*

G5. End-to-end security* for messages that are relayed through the IM server.

G6. Detection of replay attacks on clients and the IM server.

G7. Limit scopes and consequences of DoS attacks on clients and the IM server.

G8. Limit the propagation of IM worms.

*See Table 5.3 for definitions.

Table 5.2: End-user goals in IMKE

key renewal is done by a client-update, i.e. by sending users the updated key when they attempt to log in. Clients' private keys (for decryption), session keys, and MAC keys are the short-term secrets in IMKE.

5.3 The IMKE Protocol

We describe the IMKE protocol in three phases: the password authentication and key exchange, client-server, and client-client communications. We assume that IM clients are installed with the digital certificate of the IM server (or the certificate is embedded in IM clients). We discuss the IMKE protocol messages as the protocol is described below; we defer a more specific security analysis of IMKE messages to Section 6.1.2.

A, B	Two IM users (*Alice* and *Bob* respectively).
S	The IM server.
ID_A	User ID of A (unique within the IM service domain).
P	Password shared by A and S (i.e., $P = P_{AS}$).
R_A	Random number generated by A.
$\{data\}_K$	Symmetric (secret-key) encryption of *data* using key K.
$\{data\}_{K^{-1}}$	Symmetric (secret-key) decryption of *data* using key K.
$\{data\}_{E_A}$	Asymmetric (public-key) encryption of *data* using A's public key KU_A.
$\{data\}_{D_A}$	Asymmetric (public-key) decryption of *data* using A's private key KR_A.
X, Y	Concatenation of X and Y.
K^s_{AS}	Symmetric session (encryption/decryption) key shared by A and S.
K^m_{AS}	Symmetric MAC key shared by A and S.
$[X]_{AS}$	MAC output of data X under key K^m_{AS}.
"Strong" password protocol	An attacker eavesdropping on the authentication exchanges or impersonating either end of the protocol should not be able to obtain enough information to mount a successful offline dictionary attack even if a relatively *weak* password is used in the protocol [76].
Secure communications	Communications where authentication, integrity and confidentiality are achieved.
End-to-end security	Securing messages cryptographically across all points between an originating user and the intended recipient.
Repudiation	A way to ensure that the sender of a message *can* (later) deny having sent the message. This is important for casual conversations in IM [22].
Forward secrecy	The property that the compromise of long-term keys used in a protocol does not compromise any session keys established before the compromise of the long-term key.

Table 5.3: Notation and terminology used in IMKE

5.3.1 Password Authentication and Key Exchange (PAKE)

The goals that a client and a server achieve in this phase are: authenticate each other using the shared password P in a way that is resistant to offline dictionary attacks; establish a secret session key; and transport a verified dynamic public key from a client to the server. The login process between client A and server S proceeds as follows:

1. Client A generates a dynamic public/private key pair (KU_A, KR_A), and a random symmetric key K_{AS}, and then encrypts K_{AS} with the server's public key. The server's public key is verified *online*, using e.g. the *public password* method proposed by Halevi and Krawczyk [56], whereby users verify the hash of the server public key represented in plain English words. Assume for now that f_i denotes a one-way cryptographic hash function (see further discussion below). A also calculates her password function $f_1(P)$ and symmetrically encrypts KU_A and $f_1(P)$ using the session key:[1]

$$A \rightarrow S : ID_A, \{K_{AS}\}_{E_S}, \{KU_A, f_1(P)\}_{K_{AS}} \tag{5.1}$$

2. S decrypts K_{AS} by using its private key, and uses K_{AS} to retrieve KU_A and $f_1(P)$. S also independently calculates $f_1(P)$ from A's shared password P (S looks up P using ID_A). S drops the session if its locally calculated $f_1(P)$ and the corresponding value received from message (5.1) mismatch. Otherwise, S responds with a function of P, $f_2(P)$ encrypted with K_{AS}, and a random challenge R_S encrypted with A's public key:

[1] To achieve better privacy, instead of sending ID_A in the clear, it might be encrypted using K_{AS} with minor extra cost. As noted by Halevi and Krawczyk [56], this kind of user identity protection may be important for remote authentication of mobile users.

$$A \leftarrow S : \{R_S\}_{E_A}, \{f_2(P)\}_{K_{AS}} \tag{5.2}$$

3. A decrypts R_S, calculates $f_2(P)$ independently and compares it with the corresponding value received from message (5.2) and disconnects if the two quantities are unequal. Otherwise, A responds with $f_3(R_S)$:

$$A \rightarrow S : f_3(R_S) \tag{5.3}$$

4. S independently calculates $f_3(R_S)$ and compares it with the quantity received in message (5.3). If they mismatch, S disconnects; otherwise, both A and S calculate the session key (encryption key) K^s_{AS} and MAC key K^m_{AS} in the following way:

$$K^s_{AS} = f_4(K_{AS}, R_S),\ K^m_{AS} = f_5(R_S, K_{AS}) \tag{5.4}$$

   S now indicates A a successful IM client login using a message of the form (5.8). A and S also zero out K_{AS} and R_S from the program memory. This way we avoid using authentication artifacts as session keys in later communications; this is desirable for achieving forward secrecy (see Section 6.1). Nonetheless, fully zeroing-out (i.e. erasing or *forgetting*) a memory-resident secret is not a trivial task in reality [39].

Caveats. The functions f_1 and f_2 must differ; otherwise, if an attacker can replace KU_S in A's system (assuming the client machine is compromised, and the server public key is improperly[2] verified), he can deceive A without knowing P, i.e. the attacker can make A *readily* believe that she is communicating with the legitimate IM

[2]For example, in the *public password* method [56], a user may approve a wrong sequence of English words by mistake.

server. Nevertheless, even when f_1 and f_2 differ, replacing KU_S with the attacker's public key in a user's machine enables an offline dictionary attack on P. Having different f_1 and f_2 makes the attacker's active participation in the protocol harder. Note that, the strength of IMKE does not depend on the secrecy of functions f_i.

R_S and K_{AS} must be large enough (e.g. 128-bit) to withstand an exhaustive search. A must encrypt the dynamically generated public key (KU_A) while sending it to the server in message (5.1). Otherwise the following attack may succeed. Suppose an adversary generates a new private-public key pair and is able to replace KU_A with the fraudulent public key in message (5.1); this enables the adversary to decrypt R_S in message (5.2) and send a correct reply to S in message (5.3). Hence, IMKE requires the secrecy of A's public key in the PAKE phase. This use of secret "public keys" is different than a common public-key system. Privacy of public keys has been studied extensively by Bellare et al. [9]. Public keys have been used in this way elsewhere (e.g. [20, 62]).

If message (5.1) is replayed to a server by an attacker, the attacker cannot decrypt message (5.2) without knowing A's private key and K_{AS}. If message (5.2) is replayed to A by an attacker in a separate run of the protocol, A will refuse to reply to S with message (5.3) as she will fail to decrypt $f_2(P)$ (A randomly generates K_{AS} in each run of the protocol). After A has successfully logged in to the server, A receives only messages of type (5.8) from S. Therefore, if message (5.2) is replayed to A after she logs in, A can readily detect the replay, and discard that message.

The duration of the session key (K^s_{AS}) should be set carefully. This is important for clients in an *always-connected* mode, wherein clients stay logged in to the IM service for a long period of time (e.g. days or weeks). A new session key should be negotiated after a certain period (e.g. a couple of hours) depending on the expected security level and size of the session key (e.g. a shorter period for 64-bit keys than

128-bit keys) to reduce consequences from cryptographic (e.g. brute-force) attacks on the key. To do so, A and S exchange two random values K_{AS1} and R_{S1} in the following way and generate the new session key and MAC key as before. Either A or S can begin the key renewal process. The initiator must stop sending any messages before the new keys are established.

$$A \rightarrow S : \{\{K_{AS1}\}_{E_S}\}_{K^s_{AS}}, [\{K_{AS1}\}_{E_S}]_{AS} \tag{5.5}$$

$$A \leftarrow S : \{\{R_{S1}\}_{E_A}\}_{K^s_{AS}}, [\{R_{S1}\}_{E_A}]_{AS} \tag{5.6}$$

5.3.2 Client-Server Communications

After authentication, a client and server communicate mainly for the following purposes:

1. the server sends a user's contact list, block list, profile, saved settings;
2. the client sends privacy (e.g. blocked user ID) and mode information (e.g. available, away, busy);
3. the client requests to communicate with other users (e.g. adding contacts, text messaging);
4. the server responds to the client's request regarding other users; and
5. the client sends keep-alive messages (recall Section 3.1) at a pre-negotiated rate.

To send some data, $ClientData_A$, to the server S, a client encrypts the data using the previously derived session key, K^s_{AS}. The client also calculates the MAC for $ClientData_A$ and sends:

$$A \rightarrow S : \{ClientData_A\}_{K^s_{AS}}, [ClientData_A]_{AS} \tag{5.7}$$

S decrypts $ClientData_A$ using K^s_{AS} and also verifies the corresponding MAC. This provides authentication, confidentiality and integrity for the client's data to the server. To send $ServerData$ to A, S uses the following message:

$$A \leftarrow S : \{ServerData\}_{K^s_{AS}}, [ServerData]_{AS} \tag{5.8}$$

A decrypts $ServerData$ and verifies the corresponding MAC.

5.3.3 Client-Client Communications (Direct and Relayed)

Client to client IM communications happen mainly for the following purposes: server mediated/relayed messages; and bulk data transfer (e.g. file transfer, audio/video chat). Two communicating clients must get each other's public key from the server. Then the clients establish a session key known *only* to them. If A wants to send $ClientData_A$ to B, she first sends her request to communicate with B to the server S (using message (5.7)), and then the messages below follow:

1. A and B receive the other party's current dynamic public key from S (cf. message (5.8)):

$$A \leftarrow S : \{KU_B, ID_B\}_{K^s_{AS}}, [KU_B, ID_B]_{AS} \tag{5.9}$$

$$B \leftarrow S : \{KU_A, ID_A\}_{K^s_{BS}}, [KU_A, ID_A]_{BS} \tag{5.10}$$

   Note that B and S authenticate each other and derive K^s_{BS} and K^m_{BS} in the analogous way described above for A.

2. A generates a symmetric key, K_{AB} and verifies it using a challenge-response method:

$$A \rightarrow B : \{K_{AB}\}_{E_B}, \{R_A\}_{K_{AB}} \tag{5.11}$$

$$A \leftarrow B : \{R_B\}_{E_A}, \{f_6(R_A)\}_{K_{AB}} \tag{5.12}$$

$$A \rightarrow B : f_7(R_A, R_B) \tag{5.13}$$

Then A and B derive the session key K^s_{AB} and MAC key K^m_{AB} in the following way:

$$K^s_{AB} = f_8(K_{AB}, R_B),\ K^m_{AB} = f_9(R_B, K_{AB}) \tag{5.14}$$

A and B also zero out ephemeral values R_A, R_B and K_{AB} from the program memory.

3. Now, A sends $ClientData_A$ to B:

$$A \rightarrow B : \{ClientData_A\}_{K^s_{AB}}, [ClientData_A]_{AB} \tag{5.15}$$

Caveats. Although client-to-client connection setup messages (5.11–5.13) can be exchanged directly between A and B, we suggest they be relayed through the server using messages (5.7, 5.8), – i.e. with the additional encryption and MAC – to reduce threats from DoS attacks on clients. However, while relaying the setup messages, a malicious IM server can launch a typical man-in-the-middle attack [80, p.167–169] in the following way. When A notifies S that she wants to communicate with B, S generates a public key pair for B and distributes the rogue public key to A, and vice-versa. Now S can impersonate A to B and vice-versa, and thereby view or modify messages exchanged between the users. Apparently, if users exchange the connection setup messages directly, this attack could be avoided; but, if A and B get each other's

network address for direct communication from S (which is the most usual case), then this attack is still possible. The attack is made possible by the facts that, (1) users do not share any secret between them, and (2) they do not use any authenticated (long-term) public key. Note that, this is an *active attack* where the server needs to participate in a protocol run online.

In general, IM accounts are anonymous, i.e. users can get an IM account without giving any identifying information to the server.[3] Therefore, the motivation to launch the man-in-the-middle attack against random users appears less rewarding for the server. In a public IM service, if the server launches this attack against all users, the attack may easily be exposed, only if a pair of users attempt to verify their (dynamically generated) public keys through e.g. a web site or another IM service. Complex methods, e.g. the *interlock* protocol [145], may also be considered to expose an intruding IM server. However, in IMKE, we trust the server to relay the correct client public keys – as it appears less likely for the server to be able to continue such attacks without being noticed. An area of potentially interesting future research could be how to reduce the trust assumptions required on the server, and yet still have an efficient relaying protocol.

Message (5.15) is used to send $ClientData_A$ directly from A to B. For relaying data through the server, the same message type can be used. As these messages are encrypted with K^s_{AB}, which is shared only between A and B, S cannot decrypt them. Hence, goal **G5** is apparently satisfied.

If message (5.11) is replayed to B by an adversary, the adversary gains no useful information from B's reply in message (5.12). In messages (5.7), (5.8) and (5.15), the receiver retrieves data and verifies the associated MAC. The first parameter of these

[3]From the IP address of a particular user, the server may be able to retrieve the user's location in many cases (e.g. [132]), and thereby associate an IM account to some (albeit indirect) identifying attributes of a real-world user.

messages provides data confidentiality and the second part ensures data integrity and data origin authentication. The second part limits DoS attacks (goal **G7**): if one party fails to verify the MAC, it ignores or drops that connection. DoS attacks are also discussed in Section 6.1.3. To detect replay attacks (goal **G6**), $ClientData_A$ and $ServerData$ are appended/prepended with time-stamps or sequence numbers, with appropriate checks by the receiver (e.g. see [101, p.417–418] and our IMKE implementation in Chapter 7). Freshly generated session keys and clients' public keys help in detecting replays from earlier protocol runs.

If two clients communicate for a long time (in a session), they may re-negotiate a session key (and a MAC key) in a similar way as described for the client-server key renewal (recall Section 5.3.1).

Chapter 6

Security and Performance Analysis of IMKE

We analyze security and performance issues related to IMKE and provide an informal threat model for IMKE. The performance comparison with similar PAKE protocols provided in this chapter is analytical. See Chapter 7 for results on the empirical execution performance of IMKE.

6.1 Security Analysis

In this section, we provide a partial BAN-like (Burrows-Abadi-Needham [30]) analysis intended to provide a baseline of confidence in the security of IMKE. The setup for our analysis, and other security properties of IMKE are also discussed. A full BAN logic analysis would be a reasonable next step. While BAN analysis is somewhat informal in certain aspects and is well-known to have shortcomings [50, 23], it is nonetheless helpful in explaining the reasonings behind security beliefs of protocol designers, and often leads to security flaws being uncovered. However, a more rigorous

security analysis as well as a "proof" of security of IMKE using alternate (non-BAN) techniques (e.g. see Bellare-Rogaway [11]; but also Koblitz and Menezes [83] for limitations of "provable security") would be preferable to provide supplementary confidence, and is the subject of future work. We thus consider the analysis in this thesis to be a first step.

6.1.1 Setup for the Analysis

Table 6.1 lists definitions used in the IMKE analysis (borrowed in part from Burrows et al. [30]). Table 6.2 lists the technical sub-goals of IMKE which are, although idealized, more concrete and specific than the end-user goals (recall Table 5.2), and are of the type which can be verified from a BAN analysis point of view. The analysis in Section 6.1.2 shows how IMKE achieves the technical sub-goals, and leading to the end-user goals G1, G2, G3, G4 and G8. We also provide the operational assumptions (Table 6.3), and an informal IM threat model for IMKE. Table 6.4 summarizes the IMKE protocol messages to facilitate our discussion.

A *believes* X	User A behaves as if X is true.
A *once said* X	User A at some past time sent a message including X.
X is *fresh*	A message X is said to be *fresh* if (with very high probability) it has not been sent in a message at any time before the current protocol execution.
A *controls* X	User A is an authority on X (she has *jurisdiction* over X) and should be trusted on this matter.

Table 6.1: BAN-like definitions used in the IMKE analysis

T1. A and S show evidence that they know the shared (secret) password P.[a]

T2. A believes that she is communicating (in real-time) with another party that knows S's private key.

T3. S believes that it is communicating (in real-time) with another party that knows A's private key.

T4. A believes that she is communicating (in real-time) with another party that knows B's private key.

T5. B believes that he is communicating (in real-time) with another party that knows A's private key.

T6. A and S believe that they share a (secret) session key and a MAC key.

T7. A and B believe that they share a (secret) session key and a MAC key.

[a]See assumption A1 in Table 6.3; this goal is fulfilled when both parties demonstrate knowledge of the pre-established password P.

Table 6.2: Technical sub-goals of IMKE

IM Threat Model.

A *threat model* (e.g. Bishop [18, p.498]; see also [98]) identifies the threats a system is designed to counter, the nature of relevant classes of attackers (including their expected attack approaches and resources, e.g. techniques, tools, computational power, geographic access), as well as other environmental assumptions and conditions. Our IM threat model is not what would typically be expected of a *formalized* (academic) threat model, but it nonetheless provides a practically useful and clear definition of what types of attacks we intend that IMKE provides protection against. Table 6.5 lists the assumptions in our IM threat model.

We provide a few additional comments related to Table 6.5. Modern operating systems provide reasonable protection for process-memory spaces; yet, accessing a process's memory from the context of a compromised privileged (*root* or *administrator*) process is not difficult (e.g. [8]). Zeroing out memory-resident secrets is not easy

A1. Each IM user shares a user-chosen password only with the legitimate IM server (e.g. established *a priori* using out-of-band methods), and the password is not stored long-term on the user machine.

A2. The IM server's valid, authentic public key is known to all parties.

A3. Each party controls the private key for each public key pair they generate, i.e. the private key is not known or available to other parties.

A4. IMKE clients use fresh keys and challenge values where specified by the protocol, e.g. they do not intentionally reuse old values.

A5. The IM server relays clients' public keys correctly (i.e. without any modifications).

Table 6.3: Operational assumptions of IMKE

(see item 4 in Section 5.3.1) as well. Threats from keyloggers are also significant (see Section 5.2.1). Malicious programs can be used to control a large number of machines in a high-speed Internet environment. The availability of such a powerful computing platform increases attackers ability to challenge cryptographic primitives. Therefore, meeting the threat model assumptions in reality is not trivial. Nonetheless, these challenges are faced by many security protocols in practice.

6.1.2 Analysis of IMKE Messages

We analyze IMKE messages and their possible implications in different phases of the protocol run. Refer to the earlier protocol description (Section 5.3) for the actions each party takes upon receiving a message. We start by analyzing message $a1$ (recall the message labels in Table 6.4). Upon successful verification of $f_1(P)$ by S, the locally calculated $f_1(P)$ by S is the same as the $f_1(P)$ retrieved from $a1$. Message $a1$ thus implies the following.

1. A believes that K_{AS} and KU_A are fresh, as they are freshly generated by herself.

Phases	*Message Labels*	*Messages*
Authentication and Key Exchange	$a1$	$A \rightarrow S : ID_A, \{K_{AS}\}_{E_S}, \{KU_A, f_1(P)\}_{K_{AS}}$
	$a2$	$A \leftarrow S : \{R_S\}_{E_A}, \{f_2(P)\}_{K_{AS}}$
	$a3$	$A \rightarrow S : f_3(R_S)$
Public Key Distribution	$b1$	$A \leftarrow S : \{KU_B, ID_B\}_{K^s_{AS}}, [KU_B, ID_B]_{AS}$
	$b2$	$B \leftarrow S : \{KU_A, ID_A\}_{K^s_{BS}}, [KU_A, ID_A]_{BS}$
Session Key Transport	$c1$	$A \rightarrow B : \{K_{AB}\}_{E_B}, \{R_A\}_{K_{AB}}$
	$c2$	$A \leftarrow B : \{R_B\}_{E_A}, \{f_6(R_A)\}_{K_{AB}}$
	$c3$	$A \rightarrow B : f_7(R_A, R_B)$

Table 6.4: Summary of IMKE messages (see Table 5.3 for notation)

2. Before the protocol run, S knows that it shares P with A. Here, S gains the evidence that the keys K_{AS} and KU_A which message $a1$ links to P, were generated by and associated with A. Hence, S believes the identity of A, which partially satisfies goal **T1**.
3. S believes that A once said that K_{AS} and KU_A are fresh.
4. S believes that A has a valid copy of its public key KU_S.

The successful verification of message $a2$ means that the locally calculated $f_2(P)$ by A is the same as the $f_2(P)$ decrypted from $a2$. This implies the following.

1. A believes that S knows P, thus satisfying goal **T1**.
2. Knowing the private key KR_S enables S to decrypt K_{AS} and KU_A in message $a1$. S encrypts $f_2(P)$ using K_{AS}; hence, the successful verification of $f_2(P)$ by A implies that A is communicating (in the current protocol run) with a party that knows S's private key, thus satisfying goal **T2**.
3. A believes that the current message $a2$ is fresh as KU_A is fresh; this provides assurance to A that the current protocol run is not a replay.

M1. The IM client software is *trusted*. By *trusted* we mean the IM client software has not been tampered with and the underlying operating system protects the IM client's memory space (RAM and virtual memory) from other programs (including malicious programs). This assumption is required as ephemeral secret keys are stored in the program memory.

M2. Communications between IM servers are secure using e.g. encryption and message authentication. IMKE does not provide security for server-to-server messaging.

M3. Software and hardware keyloggers are not installed.

M4. Clients' keys (public/private and symmetric) stay only in program memory which are zeroed out upon terminating the program.

M5. The server public key stored in client machines is verified at each login attempt (using e.g. the *public password* method proposed by Halevi and Krawczyk [56]).

M6. Underlying communication channels need not be secure; attackers are assumed capable of viewing, altering, inserting and deleting any bitstream transferred from IM clients or servers.

M7. A user's computing/communication device need not be secure, in the following sense: attackers can put any malcode in the system or change system's settings (e.g. registry and DNS entries), provided that assumptions A2 (see Table 6.3), M1, M3 and M5 are met.

M8. We consider *realistic attackers* [56] who can exhaustively search over a password dictionary (e.g. 2^{64} computational steps) but cannot defeat (in a reasonable amount of time) the cryptographic primitives (e.g. 2^{80} computational steps) used in the protocol.

Table 6.5: IM threat model assumptions

4. A believes that S once said that R_S is fresh in the current protocol run.

The successful verification of message $a3$ by S means that the locally calculated $f_3(R_S)$ by S is the same as received in $a3$. This and the login success response from S to A imply the following.

1. S receives the evidence that A knows her private key KR_A, otherwise A could not decrypt R_S in message $a2$. Hence, goal **T3** is established.
2. The current message $a3$ is fresh as R_S is fresh; this guarantees S that the current protocol run is not a replay.
3. In message $a2$, A retrieves R_S using her dynamic private key for the current protocol run. At this point only S has a copy of A's public key. Therefore from the login success message, A believes that S possesses a valid copy of A's public key KU_A.
4. As both A and S derive the session key K^s_{AS} and MAC key K^m_{AS} from their ephemeral shared secrets (K_{AS} and R_S), goal **T6** is achieved.

From messages $b1$ and $b2$, A and B get each other's public keys from S securely. In message $b1$, A receives the public key of B (KU_B) encrypted under the shared key K^s_{AS} providing confidentiality of KU_B. Also, the MAC in message $b1$ provides integrity of KU_B. Message $b2$ provides similar guarantees to B for A's public key.

The successful verification of all of messages $c1$, $c2$ and $c3$ (see Section 5.3.3) allows the following informal line of reasoning.

1. A believes that she shares K_{AB} with B, as only B could decrypt R_A in $c1$ and respond with a function of R_A in $c2$.
2. B believes that he shares K_{AB} with A, because only A knows KR_A which is necessary to recover R_B for use in message $c3$, and the chain of messages links R_B with R_A, and R_A back to K_{AB}.

3. A and B achieve some assurance of freshness through the random challenges R_A and R_B respectively.
4. A and B receive each other's public keys securely from a trusted source S (in messages $b1$ and $b2$). The successful verification of message $c2$ provides the evidence to A that B knows the private key corresponding to B's public key which A received earlier from S, thus satisfying goal **T4**. Message $c3$, when verified, provides the similar evidence to B, thus satisfying goal **T5**.
5. A and B derive the session key K^s_{AB} and the MAC key K^m_{AB} from their ephemeral shared secrets (K_{AB} and R_B), thus goal **T7** is achieved.

We now provide informal reasonings regarding how end-users' goals G1, G2, G3, G4 and G8 (recall Table 5.2) are satisfied. We argue that in our PAKE phase, it is computationally infeasible to launch offline dictionary attacks on password P (assuming our assumptions in Table 6.5 and 6.3 are not violated). To recover plaintext $f_1(P)$ from message $a1$, an attacker apparently has to guess K_{AS}, which is computationally infeasible if K_{AS} is generated from a large key space (e.g. 128-bit or more). Another way to recover plaintext $f_1(P)$ is to learn K_{AS} by guessing the server's private key. Brute-force attacks on K_{AS} or KR_S appear to be computationally infeasible if the key length is chosen appropriately. To recover plaintext $f_2(P)$ from message $a2$, an attacker must guess K_{AS}, which is infeasible. This apparently makes P resistant to presently known offline dictionary attacks. As goal T1 is fulfilled in messages $a1$ and $a2$ without exposing P to offline dictionary attacks, IMKE achieves goal **G1**. Goal T6 establishes that A and S achieve confidentiality, and integrity (with authentication) using the secret session key K^s_{AS} and the MAC key K^m_{AS} respectively. Technical sub-goal T6, along with G1, now satisfies goal **G2**.

A and B do not authenticate each other directly. They trust the other party's identity as they receive each other's public key from S and trust S on the authenticity

of those public keys. Thus fulfilling sub-goals T4, T5 and T7 provides A and B a way to communicate securely and satisfies goal **G3**.

Message authentication between A and B is achieved by MACs, instead of digital signatures. The same session and MAC keys are shared between A and B, which provide confidentiality and authentication of the messages exchanged between the users. Any message created by A can also be created by B. Therefore the sender of a message can *repudiate* generating and sending the message. Clients' public keys are also temporary, hence binding an IM identity with the real user is technically impossible. The confidentiality of communications channels between users is protected by session keys generated from random nonces, instead of users' long-term secrets; so, the exposure of long-term secrets does not compromise past session keys. Thus repudiation and forward secrecy (goal **G4**) of users' messages are achieved (for more discussion on forward secrecy, see "Exposure of Secrets" below).

By applying techniques described in Section 4.4.2 with the secure IM connections (goal G2 and G3), we can significantly limit the propagation of IM worms, and thus satisfy goal **G8**.

Hence we have provided informal sketches of how end-user goals G1, G2, G3, G4 and G8 are established.

6.1.3 Other Security Attributes of IMKE

Below we discuss a few more security attributes of IMKE. These properties make IMKE resistant to several recently devised attacks on security protocols.

Chaining of Messages.

In the PAKE phase, messages $a1$ and $a2$ are cryptographically linked by KU_A, and messages $a2$ and $a3$ are cryptographically linked by R_S. Moreover, both KU_A and

R_S are dynamically generated in each protocol run. According to Diffie et al. [40] this kind of the chaining of protocol messages may prevent *replay* and *interleaving* attacks. Abadi and Needham [1] also recommend sufficient *connection* between protocol messages.

DoS Attacks.

Significant denial of service (DoS) attacks are generally easier to launch against an IM server than IM clients as the server is required to interact with many clients. In the PAKE phase of IMKE, the server can verify the identity of a user from the first authentication message ($a1$) it receives from the user (recall Section 6.1.2). Therefore the server can terminate connection attempts from non-legitimate (perhaps malicious) users after processing only message $a1$. Because the IM server does not wait (thereby avoids allocating memory) on any non-legitimate user, IMKE helps the IM server in limiting DoS attacks that exhaust the server's memory.

However, a potential DoS attack against the server's computing resources and memory is the following: an attacker captures message $a1$ from a successful protocol run and sends numerous copies of $a1$ to the IM server. Because these messages are valid (although stale), the server replies with $a2$; the attacker is unable to reply with $a3$ and eventually after a time-out period the connection is dropped by the server, but the server has already spent resources in processing those replayed $a1$ messages. This attack is more significant when launched in a distributed manner using many valid $a1$ messages collected from different users or separate protocol runs of the same user.

Additional known techniques, e.g. puzzles [27], may be used to mitigate DoS threats on IM servers. In general, however, many DoS attacks are likely to exist, and hard to defend against.

Insider-Assisted Attacks.

If either of A or B is a rogue user[1] participating in IMKE, we need to guard against the following type of attack: A or B learns the password of the other party, and the session keys that they share with other users (except K^s_{AB}). In IMKE, users never receive a protocol message containing any element related to other users' passwords or session keys; thus, IMKE avoids these insider-assisted attacks even when IMKE assumptions are violated by malicious users.

Exposure of Secrets.

IMKE provides forward secrecy (see Table 5.3 for definition) as the disclosure of a client-server password (long-term secret keying material) does not compromise the secrecy of the exchanged session keys from protocol runs (using that password) before the exposure. Exposure of the IM server's long term private key allows an attacker to launch offline dictionary attacks on $f_1(P)$ although the attacker cannot compromise the session key or readily impersonate S. If the session key K^s_{AS} between A and S is exposed, an attacker cannot learn P. However, the disclosure of an ephemeral key K_{AS} (which is supposed to be zeroed out from the program memory after the PAKE phase) enables an offline dictionary attack on $f_1(P)$. Although the disclosure of A's dynamic private key (which exists in the program memory as long as A remains logged in[2]) enables an attacker to reply correctly in message $a3$, IMKE still provides forward secrecy.

When both the IM server's long term private key and a user's dynamic private key are exposed, an attacker can calculate the session key from the collected messages of

[1]For example, someone who, maliciously or naively, exposes his/her dynamic private key, the client-server password, or the shared session/MAC keys.

[2]Private keys may easily be extracted from memory as Shamir and van Someren [153] outlined, if the operating system allows reading the entire memory space by any program. However, we assume that such an operation is not allowed; see assumption M1 in Section 6.1.1.

a successful protocol run; in this case, the notion of forward secrecy breaks (for the targeted session).

Denning-Sacco Attack.

The Denning-Sacco attack [37] involves an intruder who attempts to find P or impersonate A to S (or vice-versa) using a compromised session key K^s_{AS}. We have already explained above why the exposure of K^s_{AS} does not allow a dictionary attack on P. Because K^s_{AS} is not used in the PAKE phase, knowledge of K^s_{AS} does not help to impersonate as A to S or vice-versa. Although we use K^s_{AS} in the key renewal phase between A and S, the exposure of K^s_{AS} does not enable an attacker to start a key renewal phase. This is because we encrypt the random quantities in a key renewal phase also with the public key of the other party.

Many-to-Many Guessing Attack.

Kwon [86] recently (Sept. 2004) described the *many-to-many guessing attack* which can be mounted on every three-pass PAKE protocol, if a protocol is not designed and implemented carefully. In this concurrent online guessing attack, an attacker exploits the wait time of the server for the third message (which Kwon assumes to be originated from the client) to verify many password guesses in a small amount of time. Although the PAKE phase of IMKE is a three-pass protocol, IMKE is not vulnerable to this attack because the IM server verifies a client's identity from the very first authentication message.

Undetectable Online Password Guessing Attack.

Ding and Horster [41] introduced the *undetectable online password guessing attack* against three-party protocols that are known to resist offline guessing attacks. Here, in

an online transaction, an attacker verifies the correctness of his/her guessed password without revealing enough information to the server (verification authority), and hence avoids detection. Ding and Horster illustrated these attacks on some variations of the LGNS protocol (e.g. [174, 52]). In IMKE, the IM server responds only to *fresh* requests whose *authenticity* the server can verify; hence, IMKE conforms to the requirements of Ding and Horster, and thereby avoids this attack.

6.2 Performance Analysis (Analytical)

In this section, we provide an analytical performance review of IMKE. Also, we provide a rough comparison of a modified version of the PAKE phase of IMKE with a few other PAKE protocols.

In a PAKE protocol run, generally the public key operations dominate the protocol's execution performance. We summarize the key generation, public and symmetric key operations of IMKE (the PAKE phase) in Table 6.6.

	Client	*Server*
generation	1 random number and 1 PK-pair	1 random number
public key	1 encryption and 1 decryption	1 encryption and 1 decryption
symmetric key	1 encryption and 1 decryption	1 encryption and 1 decryption

Table 6.6: Cryptographic operations required by IMKE in the PAKE phase

We do not expect the computation expense of public key operations to be an issue as the data being encrypted in IMKE using public keys is always small random numbers (e.g. 128-bit), which may fit into one block of any public key cryptosystem. In contrast, data being encrypted in many PAKE protocols (e.g. variants of LGNS [174, 52]; Halevi-Krawczyk [56]) using public keys may not always fit into one block;

so in actual implementations, these protocols may require multiple public key encryptions (and subsequent decryptions). After the PAKE phase, most of the cryptographic operations in IMKE require only symmetric key operations which are generally very efficient. Also, only clients generate the dynamic public keys in IMKE, saving the server from the cost of these operations (generating public keys may be expensive, e.g. in RSA). Evidence suggests that these cryptographic operations should not undermine the instant nature of IM as they are implemented and studied elsewhere (e.g. [82, 192]). More concrete cost-efficiency characteristics of IMKE are available from our implementation (see Chapter 7).

To use IMKE as a generic PAKE protocol, clients do not need to send dynamic public keys to the IM server (or to be subsequently verified by the server). In a modified IMKE protocol, a client might perform one public key encryption and the server one public key decryption. A modified PAKE phase of IMKE is given below. However, we have not analyzed this modified protocol for security properties.

$$A \rightarrow S : ID_A, \{K_{AS}\}_{E_S}, \{f_1(P)\}_{K_{AS}} \tag{6.1}$$

$$A \leftarrow S : \{R_S, f_2(P)\}_{K_{AS}} \tag{6.2}$$

$$A \rightarrow S : f_3(R_S) \tag{6.3}$$

Note that, the modified IMKE protocol does not provide forward secrecy. For comparison, we consider the version of Halevi-Krawczyk protocol [56] that provides mutual authentication without forward secrecy (given below in simplified form). Assume that $MAC_K(X)$ is the MAC of X under key K.

$$A \leftarrow S : R_S, KU_S \tag{6.4}$$

$$A \rightarrow S : ID_A, R_S, \{K_{AS}, f_1(P, R_S, K_{AS}, ID_A, ID_S)\}_{E_S} \tag{6.5}$$

$$A \leftarrow S : MAC_{K_{AS}}(R_S, ID_S, ID_A) \tag{6.6}$$

Another similar protocol[3] is the basic Kwon-Song two-party protocol [87]. All these protocols require one public key encryption and one decryption. However, each public key operation may require multiple steps depending on the data block size. Data encrypted (or decrypted) in the Halevi-Krawczyk and Kwon-Song protocols may exceed the block size, which will increase the cost of the public key operations. As noted earlier, IMKE always performs public key operations only on random quantities which appear to fit in a single block size for all public key cryptosystems.

[3] i.e. a protocol that provides mutual authentication, using a known public key, but does not provide forward secrecy. Forward secrecy can be added to such a protocol by incorporating a Diffie-Hellman exchange.

Chapter 7

Implementation of IMKE

We have implemented IMKE using the open source Jabber[1] server and client on the Linux operating system. We chose `jabberd2` [128] as our Jabber server platform and `Gaim` [125] as our Jabber IM client. We used `OpenSSL` [130] for our general purpose cryptography library and `MySQL` [129] for database support.

One reason for choosing the Jabber (XMPP) protocol for our target implementation is its openness. Another reason is that Jabber supports the SASL (Simple Authentication and Security Layer) [111] protocol, which can accommodate a wide-range of authentication protocols. We simply added IMKE as another mechanism for authentication and key exchange while keeping the existing Jabber mechanisms (e.g. `PLAIN`, `DIGEST-MD5`) available. As Jabber is a distributed IM service (Jabber servers are controlled by different organizations; see Section 1.2), we see Jabber as the most natural platform to deploy IMKE incrementally.

In this chapter, we discuss cryptosystems and parameters that we used in the IMKE implementation. The authentication message flow of the IMKE-enabled Jabber protocol is listed, and deployment and usability issues of IMKE are discussed.

[1] http://www.jabber.org

Empirical measures for performance analysis are calculated. Lessons learned from the implementation process are listed as well.

7.1 Protocol Specification

This section outlines the implementation choices that we made while integrating IMKE with Jabber. Choices for the cryptographic library, public/symmetric cryptosystems, cryptographic hash functions, and the source of randomness are discussed.

Table 7.1 summarizes cryptosystems and parameters for the implementation of IMKE using OpenSSL. IMKE requires a long-term public key for the IM server and a dynamically generated public key for each client in every login attempt. The server public key may need to remain valid for years; however, a client's public key may live from only a few seconds to a few days, depending on how long a user remains connected per login. Therefore, the server public-key size should be a standard one and the client public key can be relatively small in size. We chose 2048-bit and 1024-bit RSA public keys for the IM server and IM clients respectively.

We used hash functions in a *double-hashing* mode; i.e. the output of a hash function is again used as input to the hash function to reduce any possibility of length extension attacks [64, 94]. For random nonce and key generation, we used the cryptographically secure pseudo-random number generator (PRNG) built in the OpenSSL library. We seeded the OpenSSL PRNG with 1024 bytes from the `/dev/urandom` file. Although `/dev/random` is a better source of randomness than `/dev/urandom` – at least theoretically [91] – we used `/dev/urandom` to seed the OpenSSL PRNG for better responsiveness.

The EME-OAEP padding (defined in PKCS #1 v2.0 [78]) was used for the RSA encryption and decryption. While generating RSA keys, we used 65537 as the public

exponent (e). We also enabled the RSA *blinding* in OpenSSL to prevent *timing attacks* [29] before performing any RSA-key operations. We do the `base64` [21] encoding (and appropriate decoding) of all binary messages (e.g. encryption outputs) to conform with the Jabber specifications; however, the base64 encoding increases the outgoing message size by about 33%. We encode the lengths of different fields in a message by converting each field's integer length into characters (2 bytes). The receiver of a message checks whether the length of the received message conforms to the sum of encoded field-lengths. A connection is dropped if the length-encoding differs.

Public key encryption	`RSA` [78] 1024/2048-bit key
Symmetric encryption	`AES-128` [113] (CBC mode)
Hash functions	`SHA-1` [112], `RIPEMD-160` [138] (160-bit output)
MAC functions	`HMAC` [84] using `SHA-1`
Source of randomness	`/dev/urandom`

Table 7.1: Cryptosystems and parameters for the IMKE implementation

We used `SHA-1` for f_1, f_3, f_4, f_6, f_8 and `RIPEMD-160` for f_2, f_5, f_7, f_9 (see Section 5.3) and all the hash functions give 20-byte (160-bit) output. We assume each client has an authentic copy of the server's RSA public key (n_S, e_S) extracted from a local file. Before a client starts the authentication phase, the server sends the list of supported SASL mechanisms. Our modified Jabber server offers `IMKE` along with the existing mechanisms. The IMKE-enabled Gaim client chooses IMKE by default and follows the authentication messages as described in Section 7.2. However, we introduced a command line option (`xmpp`) to force the client to use the standard XMPP authentication (e.g. DIGEST-MD5). Assuming user A's RSA public key is (n_A, e_A), the IMKE PAKE phase (recall Section 5.3.1) is instantiated in the following way.

$$A \rightarrow S : ID_A, \{K_{AS}\}_{E_S}, \{n_A, e_A, f_1(P)\}_{K_{AS}} \tag{7.1}$$

$$A \leftarrow S : \{R_S\}_{E_A}, \{f_2(P)\}_{K_{AS}} \tag{7.2}$$

$$A \rightarrow S : f_3(R_S) \tag{7.3}$$

We use a 4-byte sequence number with every message to foil replay attacks. After authentication, both the client and the server maintain a pair of "send-sequence" and "receive-sequence" numbers. The send-sequence is incremented before sending a message and the receive-sequence is incremented after successfully receiving a message. If the sequence number in an incoming message mismatches with the local receive-sequence, the receiver (a client or server) of that message will disconnect. The first two bytes of the `RIPEMD-160` hash output of the client-server MAC key (K^m_{AS}) are used to initialize the send- and receive-sequence. The send-sequence number is prepended with the $ClientData_A$ and the $ServerData$ in messages (5.7) and (5.8).

If A wants to communicate with B (e.g. A opens a window to send a text message or a file), A sends a special `init` message to S (using message (5.7)). Then S forwards this message to B appending A's RSA public key to it (using message (5.8)). B extracts A's public key and responds with another `init` message to S which S forwards to A appending B's RSA public key to it. Then, A and B exchange messages (5.11, 5.12 and 5.13) through the server and generate the session key and the MAC key.[2] A and B also initialize a pair of send- and receive-sequence numbers using the first two bytes of the `RIPEMD-160` hash output of the shared MAC key (K^m_{AB}). In fact, A will maintain a separate pair of send- and receive-sequence for each of her com-

[2]Note that, although relaying client-to-client connection setup messages through the server is not recommended to avoid man-in-the-middle attack from a malicious server, here we relay those messages through the server for the sake of simplicity of our implementation.

municating peers. If the sequence number in an incoming message contradicts with the local receive-sequence, the receiver (A or B) of that message will get a warning message from the IM client, and may decide to continue the session or not. However, in the case of a MAC mismatch for a message – in client-server or client-client communications – the connection is discontinued.

Message Encryption.

All client-server and client-client messages follow the original Jabber specifications. After the authentication phase, IMKE encrypts all outgoing messages, including the formatting information of a message. Text messages between peers are relayed through the server. A client encrypts the original text of an instant message (sending to another client through the server) using the shared key with the recipient of that message. However, information required by the server to interpret and forward an instant message to the recipient are encrypted by the key shared between the server and the sender. So a text message is effectively encrypted twice – once by the shared-key with the recipient, and again by the shared-key with the server. Although this multiple encryption could be avoided by allowing minor changes in the Jabber messaging format, we accept this relatively minor inefficiency to comply with the Jabber protocol. As symmetric encryption is quite fast, the effect of multiple encryptions is barely noticed in reality, especially for applications involving heavy use of user interfaces. Multiple encryptions are not required for the direct P2P file data transfer.

As noted in Table 7.1, we used AES in the CBC mode for symmetric encryption. We used the CBC mode for its desirable security properties (e.g. see [101, p.228–233] and [42] for more on encryption modes). However, in AES-CBC, as every message (client-server or client-client) is encrypted with the same IV (Initialization Vector) and the same shared key, identical cipher blocks could result when the same plaintext

message is encrypted. For example, the keep-alive message, "`\t`" is sent from a client to a server once in every minute in a default setting. This could potentially lead to replay attacks and help cryptanalysis in discovering the encryption key. To result in different cipher blocks even when sending the same plaintext, we prefix the send-sequence number to each plaintext message before encryption.

7.2 IMKE Authentication Message Flow in Jabber

In this section, we list the actual messages exchanged between a server and client in a successful authentication attempt to aid the understanding of how IMKE is embedded in the Jabber protocol. We kept the stream initialization messages for the Jabber protocol unchanged. For an example run of the original Jabber protocol, see the protocol RFC ([148, p.31–33]).

1. The client sends its XML version to the server (`ludlum`).

   ```
   <?xml version='1.0' ?>
   ```

2. The client initiates a stream to the server.

   ```
   <stream:stream to='ludlum' xmlns='jabber:client'
   xmlns:stream='http://etherx.jabber.org/streams' version='1.0'>
   ```

3. The server responds with a stream tag and the available authentication mechanisms to the client.

   ```
   <?xml version='1.0'?>
       <stream:stream xmlns:stream='http://etherx.jabber.org/streams'
       xmlns='jabber:client' from='ludlum'  version='1.0'
       id='vztr2pacd9rbuycr08arg69wkna5kyxgyr70hhdu'>
   <stream:features xmlns:stream='http://etherx.jabber.org/streams'>
       <mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
       <mechanism>DIGEST-MD5</mechanism>
       <mechanism>PLAIN</mechanism>
   ```

```
    <mechanism>IMKE</mechanism>
    </mechanisms>
</stream:features>
```

4. The client chooses `IMKE` and sends the base64 encoded message (5.1).

   ```
   <auth xmlns='urn:ietf:params:xml:ns:xmpp-sasl' mechanism='IMKE'>
   ```

   [base64 encoded message]

   ```
   </auth>
   ```

5. The server responds with the base64 encoded message (5.2) as a `challenge` message.

   ```
   <challenge xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
   ```

   [base64 encoded message]

   ```
   </challenge>
   ```

6. The client responds to the challenge with the base64 encoded message (5.3).

   ```
   <response xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
   ```

   [base64 encoded message]

   ```
   </response>
   ```

7. The server informs the client of a successful login.

   ```
   <success xmlns='urn:ietf:params:xml:ns:xmpp-sasl'/>
   ```

8. The client initiates a new stream to the server.

   ```
   <stream:stream to='ludlum' xmlns='jabber:client'
           xmlns:stream='http://etherx.jabber.org/streams'
           version='1.0'>
   ```

9. The responds by sending a stream to the client.

   ```
   <?xml version='1.0'?><stream:stream
           xmlns:stream='http://etherx.jabber.org/streams'
           xmlns='jabber:client' from='ludlum' version='1.0'
           id='q46l3ic3qnu7t44esekcchsb90bqhol8trtwa601'>
   ```

7.3 Empirical Performance, Usability, and Deployment

In this section, we discuss the empirical performance analysis, usability, and incremental deployment issues of IMKE.

Empirical Performance Analysis.

We tested the performance of IMKE in two different settings. By separating the IMKE implementation-specific code (both for the server and the client), we made a performance test client from that code to measure running time of the protocol. The actual running time of the protocol as in the ordinary Gaim client was also measured. The IMKE-enabled Jabber server was run on an `IBM xSeries 345` server and the Gaim clients, as well as the test clients, were run from `IBM IntelliStation M Pro` workstations under Linux; Table 7.2 provides the specific configuration of our test setup.

The execution time of the PAKE phase (messages $a1$, $a2$ and $a3$; see Table 7.3[3]) and the client-to-client communication key setup phase (messages $c1$, $c2$ and $c3$) were measured by the test client (excluding latency). We were interested in these phases as they require *expensive* public key operations. We ran both phases 1000 times each and took the average running time. Table 7.4 lists the execution time of IMKE messages in milliseconds ($msec$). The execution time of the PAKE phase is divided into the time spent by the client and server in Table 7.5.

The login time, the client-to-client encryption and MAC key setup time, and the file transfer data rate (using 4096-byte block size in a 100Mbps LAN) in the Gaim client were measured for IMKE and XMPP (DIGEST-MD5 authentication)

[3]Here we repeat Table 6.4 for convenience.

	Server (`IBM xSeries 345`)	*Workstation* (`IBM IntelliStation M Pro`)
Processor	1 x Intel® Xeon™ CPU 2.40GHz	1 x Intel® Pentium® 4 CPU 2.80GHz
L2 Cache	512KB	512KB
System bus	400MHz	533MHz
Memory	1GB PC2100 DDR-SDRAM	1GB PC2100 DDR-SDRAM
Chipset	ServerWorks Grand Champion™ LE	Intel E7205
Network adapter	Intel 82546EB	Broadcom BCM5702X
Operating system	Linux version 2.6.8.1	Linux version 2.6.8.1

Table 7.2: Test setup: machine configuration

and compared in Table 7.6. The clients were run manually (around 20 times) for each of the IMKE and XMPP protocols and measures were taken. The login time includes executing all the steps in the XMPP authentication (see Section 7.2), plus getting the contact list (containing only three entries) from the IM server. The client-to-client key setup phase includes the public key distribution (messages $b1$ and $b2$) time by the server as well as the actual key setup time (messages $c1$, $c2$ and $c3$). In this phase, both the clients (communication initiator and responder) required almost equal time, and in Table 7.6, we list the time required for the initiator.

The client-client or client-server message encryption and decryption, including the generation and verification of MAC and sequence number, take negligible time as they require only symmetric key operations. Our test client encrypts at 7.2MB/sec and decrypts at 4.6MB/sec (using 100-byte block size).

Phases	*Message Labels*	*Messages*
Authentication and Key Exchange	$a1$	$A \rightarrow S : ID_A, \{K_{AS}\}_{E_S}, \{KU_A, f_1(P)\}_{K_{AS}}$
	$a2$	$A \leftarrow S : \{R_S\}_{E_A}, \{f_2(P)\}_{K_{AS}}$
	$a3$	$A \rightarrow S : f_3(R_S)$
Public Key Distribution	$b1$	$A \leftarrow S : \{KU_B, ID_B\}_{K^s_{AS}}, [KU_B, ID_B]_{AS}$
	$b2$	$B \leftarrow S : \{KU_A, ID_A\}_{K^s_{BS}}, [KU_A, ID_A]_{BS}$
Session Key Transport	$c1$	$A \rightarrow B : \{K_{AB}\}_{E_B}, \{R_A\}_{K_{AB}}$
	$c2$	$A \leftarrow B : \{R_B\}_{E_A}, \{f_6(R_A)\}_{K_{AB}}$
	$c3$	$A \rightarrow B : f_7(R_A, R_B)$

Table 7.3: Summary of IMKE messages (repeated)

Usability Issues.

Our IMKE-Gaim client does not support "Remember password". We removed this option from the Gaim user-interface because malware can use a stored password (from a *predictable* disk location) to impersonate the user. Therefore, a user must (manually) input the password on every login attempt.

The PAKE phase of IMKE is more computationally intensive than the regular XMPP authentication. An IMKE client needs one public key pair generation, one public key encryption (with an RSA 2048-bit key) and one public key decryption (with an RSA 1024-bit key). In contrast, an XMPP client requires to compute only an MD5 [144] hash. Table 7.6 shows the IMKE PAKE phase takes almost twice as much time as the XMPP authentication. However, as the IMKE authentication takes less than half a second, a user barely notices any difference.

When a user initiates an IM session (e.g. text-messaging) with another user by opening a chat window, XMPP requires no extra step. In contrast, the client public key distribution and client-to-client key setup phases take place in IMKE before any

Operation	*Performed By*	*Time (msec)*
$a1$ (generation)	Client	175.98
$a1$ (processing) and $a2$ (generation)	Server	25.66
$a2$ (processing) and $a3$ (generation)	Client	4.27
$a3$ (processing)	Server	0.06
C-C key setup ($c1$, $c2$, and $c3$)	Two clients	9.13

Table 7.4: Message execution time (in milliseconds) of IMKE (test client)

	Client Time	Server Time	Total
msec	180.25	25.72	205.99
%	87.5	12.5	100

Table 7.5: Division of the PAKE phase execution time (test client)

real communications occur (text-message or file transfer) between users. The client-to-client key setup is a three-step phase which performs computationally expensive public key operations (see Section 5.3.3). However, both these phases require only 73 *msec* (see Table 7.6) of the chat initiator's time, i.e. a user can initiate more than 13 conversations per second. Also, it takes time to display the chat window by the OS, and users need time to stroke the keyboard after opening a chat window. Therefore, users do not notice any delay while starting a conversation or a file transfer. The

	Login (*msec*)	C-C key setup (*msec*)	File transfer (MB/sec)
XMPP	212	–	5.67
IMKE	393	73	5.57

Table 7.6: Comparison of the XMPP and IMKE Gaim implementations

file transfer data rate for IMKE and XMPP is comparable as in Table 7.6, and we only allow a file transfer between two IMKE users after they have completed the client-to-client key setup phase.

Other than typing the password on every login, IMKE users do not experience any other differences in our implementation. In the case of deploying IMKE incrementally, IMKE users are notified when they communicate with non-IMKE users ("Not Encrypted" in the chat window title; a lock/unlock icon, or another *user friendly* visual feedback, may be more appropriate). The IMKE user-status (whether a user is IMKE-enabled or not) is sent for every online contact in a user's contact list. A user-info window shows the protocol used by that user: "Jabber with IMKE" or "Jabber without IMKE".

We have not implemented any online verification method for the server public key. These techniques (e.g. the public password [56] method) may impose an extra step to the users while logging in. Methods introduced by Pinkas and Sander [137] (see also [157]) can reduce such usability drawbacks by using a *secure cookie*.

Incremental Deployment.

To achieve better security, it is always desirable to have all users of a system using its latest version. However, in reality it is a difficult goal to satisfy; especially for IM systems – strict enforcement may deter casual users. Our implementation of IMKE in the Jabber protocol can coexist with mainstream Jabber implementations. The IMKE-enabled Jabber server handles IMKE clients as well as standard Jabber clients. Also communications (e.g. text-messaging, file transfer) between IMKE and standard Jabber users are possible. The communication channel between an IMKE client and the IMKE-enabled Jabber server is encrypted, while the communication channel between a standard Jabber client and the IMKE-enabled Jabber server is

plaintext. So messages from an IMKE user to a standard Jabber user are encrypted for one client-to-server leg of the client-server-client two-leg trip. However, direct P2P data transfer (e.g. file transfer) between an IMKE user to a standard Jabber user is completely plaintext.

7.4 Lessons Learned

This section discusses the lessons we learned from integrating IMKE with Jabber. It provides insights on how IMKE may be practically embedded in public IM systems.

To increase efficiency, it is always desired that an authentication protocol would use the minimum number of messages (generally two to four) to achieve the protocol's goals. As it is evident in the XMPP implementation, real-life protocols take significantly more steps (see Section 7.2) and still provide a usable performance. An online server public key verification method can easily be implemented by using the extra steps in XMPP. For example, Halevi and Krawczyk's [56] method for verifying a server public key by plain English words (public password) can be implemented using the message where the IM server sends available authentication mechanisms to IM clients (i.e. message 3 in Section 7.2). Online server public key verification reduces the threat of a malicious program changing a user's local DNS cache, or the IM server's locally stored (in a user's machine) public key.

Our implementation used a fixed set of cryptosystems and parameters (see Table 7.1). Piggybacking onto existing XMPP messages, the support for negotiating public/symmetric key encryption systems as well as MAC functions can be provided to the communicating parties (client-server or client-client) without introducing any extra message.

In the PAKE phase of IMKE, both the server and a client perform one public key encryption and decryption each; in addition, the client generates a public key per login. It is well-known that the RSA public key generation is significantly more expensive than RSA encryption/decryption operations. Table 7.5 shows that the IM server does only 12.5% of the computation required in the PAKE phase. This is desirable for a typical IM setup, because the server must handle a large number of users (with limited resources) while users' machines generally remain under-utilized. However, when using IM from a (computationally) low-powered hand-held device, a public key cryptosystem with cheap key generation (e.g. ElGamal) would be more appropriate.

Using sequence numbers in the AES encryption is required to stop replay attacks as well as to reduce cryptanalysis of identical cipher blocks resulting from the same plaintext messages (see Section 7.1). We could use AES-CTR (AES in the Counter mode; see [42]) to get different cipher blocks when sending the same plaintext message. However, the OpenSSL (version 0.9.7e) that we used does not directly implement[4] AES-CTR, and AES-CBC with sequence number appears well-suited and more efficient than AES-CTR in IMKE.

[4]There is an OpenSSL-based AES-CTR implementation by Viega et al. [180, p.189-192].

Chapter 8

Conclusions and Future Work

In this chapter, we summarize the risks associated with using public IM systems, and the notable features of our proposed Instant Messaging Key Exchange (IMKE) protocol for IM. Possible improvements of IMKE, which are subject to future work, are also discussed.

Risks from IM Systems.

We have presented a survey of threats to public IM systems. IM exploits and vulnerabilities are currently making the headlines of many technical news magazines. Nevertheless, the number of IM users is rapidly increasing as well as the range of IM features. IM now offers reasonable quality audio and video, and is being used as a platform for online games. These features are attracting more new users, and encouraging the existing users to spend more time on IM. The power of IM is slowly being recognized in the business world, leading to a high penetration rate of public IM services in corporate settings. For home users, IM means *instant communication*, and for business users, IM makes *instant collaboration* a reality. The sustained growth of IM networks is bound to attract an increasing number of malcode writers

and phishers. A worm spread in IM networks may have significant impacts on other Internet services because of the large number of end-users connected by IM networks. Surprisingly, as our survey shows, there has not been much academic research on the potential threats to IM. This might be attributed to the misconceptions that IM is used only by young-adults, and that it is largely used for *gossiping* or *goofing off* (e.g. [59, 182]).

As noted earlier, relying solely on SSL-based solutions (the most common security attribute of corporate IM systems) for security in a public IM service has major limitations. For example, the SSL model allows viewing plaintext messages of users' conversations at the IM server, and assumes users' machines are completely *trusted*. Even for business IM users, SSL cannot address existing IM threats, simply because SSL is designed to address a different threat model. Several client-side (security) software plug-ins have been designed for public IM services with little improved protection against real risks.

Remarks on IMKE.

Here we review the noteworthy features of IMKE.

Although most public IM protocols are *insecure*, IM service providers are reluctant to change their IM protocols to address security issues. We attribute this inertia to two basic reasons – the security risks from IM are not well-understood, and service providers are unwilling to abandon a *mature* IM protocol in use. Therefore, we proposed IMKE – a lightweight and efficient security protocol – to enhance IM security and to reduce IM security risks without introducing major incompatibilities with existing IM protocols. Our implementation with the standard Jabber IM protocol provides evidence that IMKE can be incrementally integrated in public IM protocols without a large implementation effort.

A significant number of machines connected to the Internet are infected with malicious programs. To address this new reality, we took a step forward in the IMKE design – we designed IMKE to work even in a malicious environment with *realistic* restrictions (recall Table 6.3 and 6.5). Only a few existing Password Authentication and Key Exchange (PAKE) protocols address the problem of running a protocol in the presence of malicious programs (e.g. [185, 10]). We argue that with the prevalence of malware in end-user machines, network protocol designers must consider design choices to mitigate risks due to such malicious programs.

IMKE enables private and secure communications between two users who share no authentication tokens, mediated by a server on the Internet. The session key used for message encryption in IMKE is derived from short-lived *fresh* secrets, instead of any long-term secrets. This provides the confidence of forward secrecy to IMKE users. IMKE allows authentication of exchanged messages between two parties, and the sender is able to repudiate a message. Note that, repudiability of instant messages in a (real-life) conversation-style public IM environment is critical [22]. Also, IMKE users require no hardware tokens or long-term user public keys to log in to the IM server. The protocol provides *strong* authentication by using a memorable password and a long-term server public key. IMKE may be used to enable private and secure communications in many server-mediated three-party systems.

Future Work.

Group-chat and chat-room (recall Table 1.1 for definitions) are heavily used features in IM. A future version of IMKE would ideally accommodate these features. Introducing methods to ensure human-in-the-loop during login, e.g. challenging with a CAPTCHA [169], can stop automated impersonation using stolen/compromised user name and password. However, deploying such a method for IM networks may put an enormous

load on IM servers with millions of online users. Measures as outlined by Pinkas and Sander [137] (see also [157]) can help minimize the load on servers. Our IMKE implementation would ideally provide this feature, as well as an online server public key verification method (e.g. public password [56]) in the future.

We have measured the empirical running time for IMKE-enabled Gaim [125] clients. However, the complementary techniques that we proposed in Section 4.4.2 have not been implemented and tested; doing so may help in understanding the effectiveness of those techniques as well as their usability impacts in reality.

While we described our protocol in Section 5.3, we also discussed security caveats of the protocol. In Chapter 6 we theoretically analyzed IMKE, using a BAN-like technique. Although informal, our analysis is quite extensive, and provides baseline reasonings of confidence in IMKE. Nonetheless, a full analysis using BAN or similar formal analysis tools, as well as a "proof" of security of IMKE using non-BAN techniques would offer increased confidence in IMKE.

Concluding Remarks.

We have explored the security issues related to public IM services, proposed a security protocol for IM called IMKE, and two complementary user-friendly techniques to restrict the propagation of IM worms. We also embedded IMKE in the Jabber protocol and measured the execution performance. This allowed us to evaluate the feasibility of integrating IMKE with a popular IM protocol; the implementation effort required was moderate.

Designing a secure IM system requires serious consideration of typical end-users who use it as a casual system without being aware of the underlying threats (indeed, most users never want to be aware of the dangers of underlying software system). An overly restrictive model (i.e. with negative human interface aspects) may deter IM

users, having adverse effects, e.g. users may move to a less secure model or it may even harm the spontaneity of IM. Nonetheless, we strongly believe that security issues in IM require greater attention from the security research community, lest IM becomes as big a security problem as email, which remains the number one breeding ground for worms, despite ubiquitous security measures [81]. Our survey of IM threats, the proposed security protocol, and the implementation of the protocol help to further research on IM security.

Bibliography

[1] M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, 1996.

[2] R. Albert, H. Jeong, and A.-L. Barabási. Diameter of the World Wide Web. *Nature*, 401:130–131, 1999.

[3] America Online, Inc. AOL Instant Messenger. `http://www.aim.com/` [Accessed: Mar. 22, 2005].

[4] R. J. Anderson and S. Vaudenay. Minding your p's and q's. In *Advances in Cryptology - Asiacrypt '96*, volume 1163 of *LNCS*, pages 26–35. Springer-Verlag, 1996.

[5] Anti-Phishing Working Group. Phishing activity trends report, Feb. 2005. `http://www.antiphishing.org/APWG_Phishing_Activity_Report_Feb05.pdf` [Accessed: Mar. 31, 2005].

[6] A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.

[7] A.-L. Barabási and E. Bonabeau. Scale-free networks. *Scientific American*, 288:60–69, 2003.

[8] R. Battistoni, E. Gabrielli, and L. V. Mancini. A host intrusion prevention system for Windows operating systems. In *Proceedings of the 9th European Symposium on Research Computer Security (ESORICS 2004)*, pages 352–368, Sept. 2004.

[9] M. Bellare, A. Boldyreva, A. Desai, and D. Pointcheval. Key-privacy in public-key encryption. In *Advances in Cryptology - Asiacrypt 2001*, volume 2248 of *LNCS*. Springer-Verlag, 2001.

[10] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *Advances in Cryptology - Eurocrypt 2000*, volume 1807 of *LNCS*, pages 139–155. Springer-Verlag, May 2000.

[11] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Advances in Cryptology - Crypto '93*, volume 773 of *LNCS*, pages 232–249. Springer-Verlag, 1994.

[12] M. Bellare and P. Rogaway. Provably secure session key distribution: the three party case. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing (STOC '95)*, pages 57–66. ACM Press, 1995.

[13] M. Bellare and P. Rogaway. The AuthA protocol for password-based authenticated key exchange. Contribution to the IEEE P1363 study group, Mar. 2000. `http://grouper.ieee.org/groups/1363/passwdPK/contributions.html` [Accessed: Oct. 05, 2004].

[14] S. M. Bellovin. Security problems in the TCP/IP protocol suite. *SIGCOMM Computer Communication Review*, 19(2):32–48, 1989.

[15] S. M. Bellovin and M. Merritt. Augmented Encrypted Key Exchange: a password-based protocol secure against dictionary attacks and password file compromise. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 244–250, Fairfax, Virginia, USA, 1993. ACM Press.

[16] S. M. Bellovin and M. J. Merritt. Encrypted Key Exchange: Password-based protocols secure against dictionary attacks. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 72–84, May 1992.

[17] S. M. Bellovin and M. J. Merritt. Cryptographic protocol for secure communications. U.S. Patent 5,241,599, Aug. 1993.

[18] M. Bishop. *Computer Security: Art and Science*. Addison Wesley Professional, 2003.

[19] K. Bodine and M. Pignol. Kinetic typography-based Instant Messaging. In *CHI '03 extended abstracts on Human Factors in Computer Systems*, pages 914–915, Ft. Lauderdale, FL, USA, 2003. ACM Press.

[20] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *Advances in Cryptology - Eurocrypt 2004*, volume 3027 of *LNCS*, pages 506–522. Springer-Verlag, May 2004.

[21] N. Borenstein and N. Freed. MIME (Multipurpose Internet Mail Extensions) part one: Mechanisms for specifying and describing the format of Internet message bodies, Sept. 1993. RFC 1521, Status: Standards Track. `http://www.ietf.org/rfc/rfc1521.txt` [Accessed: Mar. 3, 2005].

[22] N. Borisov, I. Goldberg, and E. Brewer. Off-the-record communication, or, why not to use PGP. In *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society, WPES '04*, pages 77–84. ACM Press, 2004.

[23] C. Boyd and W. Mao. On a limitation of BAN logic. In *Advances in Cryptology - Eurocrypt 1993*, volume 765 of *LNCS*, pages 240–247. Springer-Verlag, May 1993.

[24] C. Boyd and A. Mathuria. *Protocols for Authentication and Key Establishment*. Springer-Verlag, 2003.

[25] C. Boyd, P. Montague, and K. Q. Nguyen. Elliptic curve based password authenticated key exchange protocols. In *Proceedings of the 6th Australasian Conference on Information Security and Privacy*, pages 487–501. Springer-Verlag, 2001.

[26] V. Boyko, P. MacKenzie, and S. Patel. Provably secure password-authenticated key exchange using Diffie-Hellman. In *Advances in Cryptology - Eurocrypt 2000*, volume 1807 of *LNCS*, pages 156–171. Springer-Verlag, May 2000.

[27] E. Bresson, O. Chevassut, and D. Pointcheval. New security results on Encrypted Key Exchange. In *Proceedings of the 7th International Workshop on Theory and Practice in Public Key Cryptography (PKC 2004)*, volume 2947 of *LNCS*, pages 145–158. Springer-Verlag, 2004.

[28] D. R. L. Brown and R. P. Gallant. The static Diffie-Hellman problem. Cryptology ePrint Archive, Report 2004/306, Nov. 2004. `http://eprint.iacr.org/2004/306` [Accessed: Apr 6, 2005].

[29] D. Brumley and D. Boneh. Remote timing attacks are practical. In *Proceedings of the 12th USENIX Security Symposium*, pages 1–14, Aug. 2003.

[30] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. In *Proceedings of the 12th ACM Symposium on Operating Systems Principles*, pages 1–13. ACM Press, 1989. `http://www.hpl.hp.com/techreports/Compaq-DEC/SRC-RR-39.pdf` [Accessed: May 25, 2004].

[31] B. Campbell, J. Rosenberg, H. Schulzrinne, C. Huitema, and D. Gurle. Session Initiation Protocol (SIP) extension for Instant Messaging, Dec. 2002. RFC 3428, Status: Proposed Standard. `http://www.faqs.org/rfcs/rfc3428.html` [Accessed: June 22, 2004].

[32] Cerulean Studios. Trillian. `http://www.ceruleanstudios.com/` [Accessed: Mar. 17, 2005].

[33] S. M. Cherry. IM means business. *IEEE Spectrum Online*, 39:28–32, Nov. 2002.

[34] N. Chou, R. Ledesma, Y. Teraguchi, and J. C. Mitchell. Client-side defense against Web-based identity theft. In *Network and Distributed System Security Symposium Conference Proceedings: 2004*, San Diego, CA, USA, Feb. 2004. The Internet Society.

[35] M. Debbabi and M. Rahman. The war of presence and Instant Messaging: Right protocols and APIs. In *1st IEEE Consumer Communications and Networking Conference*, pages 341–346, Las Vegas, NV, USA, Jan. 2004.

[36] C. A. DellaFera, M. W. Eichin, R. S. French, D. C. Jedlinsky, J. T. Kohl, and W. E. Sommerfeld. The Zephyr notification service. In *Proceedings of the USENIX Technical Conference*, pages 213–220, Dallas, TX, USA, Feb. 1988.

[37] D. E. Denning and G. M. Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):533–536, 1981.

[38] Z. Dezsö and A.-L. Barabási. Halting viruses in scale-free networks. *Physical Review E*, 65(5), May 2002.

[39] G. Di Crescenzo, N. Ferguson, R. Impagliazzo, and M. Jakobsson. How to forget a secret. In *Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science (STACS '99)*, volume 1563 of *LNCS*, pages 500–509. Springer-Verlag, 1999.

[40] W. Diffie, P. C. van Oorschot, and M. J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes and Cryptography*, 2(2):107–125, 1992.

[41] Y. Ding and P. Horster. Undetectable on-line password guessing attacks. *ACM Operating Systems Review, SIGOPS*, 29(4):77–86, 1995.

[42] M. Dworkin. Recommendation for block cipher modes of operation: Methods and techniques, Dec. 2001. NIST Special Publication 800-38A, `http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf` [Accessed: Mar. 11, 2004].

[43] EarthWeb. Instant Messaging Planet. `http://www.instantmessagingplanet.com/security/` [Accessed: Mar. 23, 2005].

[44] ElcomSoft. Advanced instant messengers password recovery. `http://www.elcomsoft.com/aimpr.html` [Accessed: Mar. 31, 2005].

[45] Eyeball Networks Inc. Eyeball Chat – Free video Instant Messenger. `http://www.eyeballchat.com` [Accessed: June 15, 2005].

[46] G. Faulhaber. Network effects and merger analysis: Instant Messaging and the AOL-Time Warner case. *Telecommunications Policy*, 26:311–333, 2002.

[47] D. Frase. The instant message menace: Security problems in the enterprise and some solutions, Nov. 2001. SANS Institute. `http://www.sans.org/rr/papers/60/479.pdf` [Accessed: Dec. 7, 2003].

[48] A. Fritzler. AIM/Oscar protocol specification, 2000. `http://aimdoc.sourceforge.net/OSCARdoc/` [Accessed: Feb. 19, 2004].

[49] A. Ghavam, R. Liscano, M. Barbeau, T. Gray, and N. D. Georganas. Secure presence-based services. In *Conference on Computer Supported Cooperative Work (CSCW 2002)*, Nov. 2002. `http://citeseer.nj.nec.com/591194.html` [Accessed: Dec. 7, 2003].

[50] V. D. Gligor, R. Kailar, S. G. Stubblebine, and L. Gong. Logics for cryptographic protocols - virtues and limitations. In *Proceedings of the IEEE Computer Security Foundations Workshop IV*, pages 219–226, 1991.

[51] P. Godefroid, J. D. Herbsleb, L. J. Jagadeesan, and D. Li. Ensuring privacy in presence awareness systems: An automated verification approach, 2000.

[52] L. Gong. Optimal authentication protocols resistant to password guessing attacks. In *Proceedings of the 8th IEEE Computer Security Foundations Workshop(CSFW '95)*, pages 24–29, 1995.

[53] 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.

[54] A. Gostev. Malware evolution: January - March 2005. Kaspersky Lab. `http://www.viruslist.com/en/analysis` [Accessed: Apr. 27, 2005].

[55] R. E. Grinter and L. Palen. Instant Messaging in teen life. In *Proceedings of the 2002 ACM Conference on Computer Supported Cooperative Work*, pages 21–30. ACM Press, 2002.

[56] S. Halevi and H. Krawczyk. Public-key cryptography and password protocols. *ACM Transactions on Information and Systems Security*, 2(3):230–268, 1999.

[57] M. Handel and J. D. Herbsleb. What is chat doing in the workplace? In *Proceedings of the 2002 ACM Conference on Computer Supported Cooperative Work*, pages 1–10, New Orleans, LA, USA, 2002. ACM Press.

[58] P. Henry and H. Luo. Off-the-record email system. In *Proceedings of the IEEE INFOCOM*, pages 869–877, Apr. 2001.

[59] J. D. Herbsleb, D. L. Atkins, D. G. Boyer, M. Handel, and T. A. Finholt. Introducing Instant Messaging and chat in the workplace. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 171–178, Minneapolis, MN, USA, 2002. ACM Press.

[60] N. Hindocha. Threats to Instant Messaging, 2003. Symantec Security Response. `http://securityresponse.symantec.com/avcenter/reference/threats.to.instant.messaging.pdf` [Accessed: Dec. 7, 2003].

[61] N. Hindocha and E. Chien. Malicious threats and vulnerabilities in Instant Messaging. In *Virus Bulletin Conference, vb2003*, Sept. 2003.

[62] D. N. Hoover and B. N. Kausik. Software smart cards via cryptographic camouflage. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, pages 208–215, Oakland, CA, May 1999.

[63] A. D. Householder. Social engineering attacks via IRC and Instant Messaging. CERT/CC, Mar. 2002. `http://www.cert.org/incident_notes/IN-2002-03.html` [Accessed: Dec. 7, 2003].

[64] R. Housley and N. Ferguson. Security design review of the Ciphire system, July 2004. `https://www.ciphirebeta.com/cm/technology/reviews.html` [Accessed: Mar. 2, 2005].

[65] Y. H. Hwang, D. H. Yum, and P. J. Lee. EPA: An efficient password-based protocol for authenticated key exchange. In *Proceedings of the Information Security and Privacy: 8th Australasian Conference (ACISP 2003)*, volume 2727 of *LNCS*, pages 452–463. Springer-Verlag, July 2003.

[66] ICQ Inc. ICQ Pro 2003b. `http://www.icq.com/` [Accessed: Mar. 22, 2005].

[67] IEEE P1363 Working Group. P1363.2: Standard specifications for password-based public-key cryptographic techniques, Mar. 2005. IEEE P1363.2/D20 (Draft version 20), `http://grouper.ieee.org/groups/1363/passwdPK/draft.html` [Accessed: Apr. 13, 2005].

[68] IMLogic. IM Manager. `http://www.imlogic.com/products/im_manager.asp` [Accessed: Mar. 17, 2005].

[69] IMLogic. IMlogic Threat Center. `http://www.imlogic.com/im_threat_center/index.asp` [Accessed: Mar. 17, 2005].

[70] Incognito Systems. iGo Incognito. `http://www.igo-incognito.com/` [Accessed: Mar. 17, 2005].

[71] E. Isaacs, A. Walendowski, and D. Ranganathan. Mobile Instant Messaging through Hubbub. *Communications of the ACM*, 45(9):68–72, 2002.

[72] E. Isaacs, A. Walendowski, and D. Ranganthan. Hubbub: A sound-enhanced mobile instant messenger that supports awareness and opportunistic interactions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 179–186, Minneapolis, MN, USA, 2002. ACM Press.

[73] E. Isaacs, A. Walendowski, S. Whittaker, D. J. Schiano, and C. Kamm. The character, functions, and styles of Instant Messaging in the workplace. In *Proceedings of the 2002 ACM Conference on Computer Supported Cooperative Work*, pages 11–20, New Orleans, LA, USA, 2002. ACM Press.

[74] Jabber Inc. Architectural considerations for presence and Instant Messaging infrastructure: Comparing XMPP and SIP/SIMPLE, May 2004. `http://www.jabber.com/index.cgi?CONTENT_ID=55` [Accessed: June 9, 2004].

[75] D. Jablon. Research papers on password-based cryptography. `http://www.jablon.org/passwordlinks.html` [Accessed: Oct. 11, 2004].

[76] D. Jablon. Strong password-only authenticated key exchange. *SIGCOMM Computer Communication Review*, 26(5):5–26, 1996.

[77] L. Joncheray. A simple active attack against TCP. In *Proceedings of the 5th USENIX Security Symposium*, pages 7–19, June 1995.

[78] B. Kaliski and J. Staddon. PKCS #1: RSA cryptography specifications version 2.0, Oct. 1998. RFC 2437, Status: Informational. `http://www.ietf.org/rfc/rfc2437.txt` [Accessed: Mar. 2, 2005].

[79] C. Kalt. Internet Relay Chat: Architecture, Apr. 2000. RFC 2810, Status: Informational. `http://www.faqs.org/rfcs/rfc2810.html` [Accessed: June 22, 2004].

[80] C. Kaufman, R. Perlman, and M. Speciner. *Network Security: Private Communication in a Public World*. Prentice Hall PTR., second edition, 2002.

[81] D. M. Kienzle and M. C. Elder. Recent worms: A survey and trends. In *Proceedings of the 2003 ACM Workshop on Rapid Malcode*, pages 1–10, Washington, DC, USA, Oct. 2003. ACM Press.

[82] H. Kikuchi, M. Tada, and S. Nakanishi. Secure Instant Messaging protocol preserving confidentiality against administrator. In *18th International Conference on Advanced Information Networking and Applications, AINA 2004*, volume 2, pages 27–30, Fukuoka, Japan, Mar. 2004.

[83] N. Koblitz and A. Menezes. Another look at "provable security". Cryptology ePrint Archive, Report 2004/152, 2004. To appear in Journal of Cryptology. `http://eprint.iacr.org/2004/152` [Accessed: Apr. 15, 2005].

[84] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-hashing for message authentication, Feb. 1997. RFC 2104, Status: Informational. `http://www.ietf.org/rfc/rfc2104.txt` [Accessed: Mar. 2, 2005].

[85] T. Kwon. Authentication and key agreement via memorable password. Cryptology ePrint Archive, Report 2000/026, 2000. `http://eprint.iacr.org/2000/026/` [Accessed: Oct. 06, 2004].

[86] T. Kwon. Practical authenticated key agreement using passwords. In *Proceedings of the 7th International Conference on Information Security (ISC 2004)*, volume 3225 of *LNCS*, pages 1–12. Springer-Verlag, Sept. 2004.

[87] T. Kwon and J. Song. Efficient and secure password-based authentication protocols against guessing attacks. *Computer Communications*, 21(9):853–861, 1998.

[88] L. Law, A. Menezes, M. Qu, J. Solinas, and S. Vanstone. An efficient protocol for authenticated key agreement. *Designs, Codes and Cryptography*, 28(2):119–134, Mar. 2003.

[89] G. Lawton. Instant Messaging puts on a business suit. *IEEE Computer Society: Computer Magazine*, Mar. 2003. `http://www.computer.org/computer/homepage/0303/Lawton/` [Accessed: Dec. 8, 2003].

[90] C. H. Lim and P. J. Lee. A key recovery attack on discrete log-based schemes using a prime order subgroup. In *Advances in Cryptology - Crypto '97*, volume 1294 of *LNCS*, pages 249–263. Springer-Verlag, 1997.

[91] Linux man pages. Linux Programmer's Manual: random(4). `http://www.die.net/doc/linux/man/man4/random.4.html` [Accessed: May 26, 2005].

[92] T. Lomas, L. Gong, J. H. Saltzer, and R. M. Needham. Reducing risks from poorly chosen keys. *ACM Operating Systems Review, SIGOPS*, 23(5):14–18, 1989.

[93] S. Lucks. Open Key Exchange: How to defeat dictionary attacks without encrypting public keys. In *Proceedings of the 5th International Workshop on Security Protocols*, pages 79–90. Springer-Verlag, 1998.

[94] S. Lucks. Design principles for iterated hash functions. Cryptology ePrint Archive, Report 2004/253, Sept. 2004. `http://eprint.iacr.org/2004/253` [Accessed: Mar. 2, 2005].

[95] P. MacKenzie. On the security of the SPEKE password-authenticated key exchange protocol. Cryptology ePrint Archive, Report 2001/057, July 2001. `http://eprint.iacr.org/2001/057/` [Accessed: Sep. 23, 2004].

[96] P. D. MacKenzie, S. Patel, and R. Swaminathan. Password-authenticated key exchange based on RSA. In *Proceedings of the 6th International Conference on the Theory and Application of Cryptology and Information Security*, pages 599–613. Springer-Verlag, 2000.

[97] Macworld.co.uk Staff. Microsoft warns of JPEG threat, Sept. 2004. News article, `http://www.macworld.co.uk/news/` [Accessed: Sep. 15, 2004].

[98] A. Main and P. C. van Oorschot. Software protection and application security: Understanding the battleground. In *Proceedings of the State of the Art and Evolution of Computer Security and Industrial Cryptography*, LNCS (to appear). Springer-Verlag, June 2003.

[99] M. Mannan and P. C. van Oorschot. Secure public Instant Messaging: A survey. In *Proceedings of the 2nd Annual Conference on Privacy, Security and Trust (PST'04)*, pages 69–77, Fredericton, NB, Canada, Oct. 2004.

[100] M. Mannan and P. C. van Oorschot. On Instant Messaging worms, analysis and countermeasures. In *Proceedings of the 3rd ACM Workshop on Rapid Malcode (WORM 2005) (to appear)*, Fairfax, VA, USA, Nov. 2005. ACM Press.

[101] A. Menezes, P. C. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[102] Microsoft. MSN Messenger. `http://messenger.msn.com/` [Accessed: Mar. 22, 2005].

[103] Microsoft. MSN Messenger for Mac. `http://www.microsoft.com/mac/default.aspx?pid=msnmessenger` [Accessed: May 3, 2005].

[104] Microsoft. Office live communications server. `http://www.microsoft.com/office/livecomm/prodinfo/default.mspx` [Accessed: Mar. 18, 2005].

[105] Microsoft PressPass. Flirting's moved online! New research from MSN reveals millions swap IM addresses with potential dates, July 2004. `http://www.microsoft.com/presspass/press/2004/jul04/07-08FlirtingPR.asp` [Accessed: Mar. 17, 2005].

[106] Microsoft TechNet. Microsoft security bulletin MS05-009: Vulnerability in PNG processing could allow remote code execution (890261). `http://www.microsoft.com/technet/security/bulletin/MS05-009.mspx` [Accessed: Mar. 18, 2005].

[107] Microsoft TechNet. Microsoft security bulletin MS05-022: Vulnerability in MSN Messenger could lead to remote code execution (896597). `http://www.microsoft.com/technet/security/Bulletin/MS05-022.mspx` [Accessed: June 20, 2005].

[108] M. Mintz. MSN Messaging protocol description. `http://www.hypothetic.org/docs/msn/index.php` [Accessed: Dec. 7, 2003].

[109] G. Mori and J. Malik. Recognizing objects in adversarial clutter: Breaking a visual CAPTCHA. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 134–141, 2003. UC Berkeley Computer Vision Group. `http://www.cs.berkeley.edu/~mori/gimpy/gimpy.html` [Accessed: June 14, 2005].

[110] M. D. Murphy. Instant message security - Analysis of Cerulean Studios' Trillian application, June 2003. SANS Institute. `http://www.giac.org/practical/GSEC/Michael_Murphy_GSEC.pdf` [Accessed: Dec. 7, 2003].

[111] J. G. Myers. Simple authentication and security layer (SASL), Oct. 1997. RFC 2222, Status: Standards Track. `http://www.ietf.org/rfc/rfc2222.txt` [Accessed: Jun. 10, 2004].

[112] National Institute of Standards and Technology. Secure hash standard, Apr. 1995. FIPS PUB 180-1 `http://www.itl.nist.gov/fipspubs/fip180-1.htm` [Accessed: Mar. 2, 2005].

[113] National Institute of Standards and Technology. Advanced Encryption Standard (AES), Nov. 2001. FIPS PUB 197 `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf` [Accessed: Aug. 6, 2004].

[114] C. Neustaedter, S. Greenberg, and S. Carpendale. IMVis: Instant messenger visualization. In *Proceedings of the 2002 ACM on Computer Supported Cooperative Work Video Program*, pages 6–6, New Orleans, LA, USA, 2002. ACM Press.

[115] C. Newman. The one-time-password SASL mechanism, Oct. 1998. RFC 2444, Updates RFC 2222. Status: Proposed Standard. `http://www.faqs.org/rfcs/rfc2444.html` [Accessed: June 18, 2004].

[116] C. Newman. Using TLS with IMAP, POP3 and ACAP, June 1999. RFC 2595, Status: Proposed Standard. `http://www.faqs.org/rfcs/rfc2595.html` [Accessed: June 22, 2004].

[117] News.com Staff. Pop-up program reads keystrokes, steals passwords, June 2004. News article, `http://news.com.com/` [Accessed: Aug. 30, 2004].

[118] News.com Staff. Study: Consumers take cyberattacks lightly, Sept. 2004. National Cyber Security Alliance (NCSA) Perception Poll Release. News article, `http://news.com.com/` [Accessed: Oct. 05, 2004].

[119] News.com Staff. Phishers target Yahoo Messenger, Mar. 2005. News article, `http://news.com.com/` [Accessed: Mar. 27, 2005].

[120] News.com Staff. Trillian IM flaw exposed, Mar. 2005. News article, `http://news.com.com/` [Accessed: Mar. 25, 2005].

[121] News.com Staff. Worm attack forces Reuters IM offline, Apr. 2005. News article, `http://news.com.com/` [Accessed: Apr. 15, 2005].

[122] News.com Staff. Yahoo fills in Messenger hole, May 2005. News article, `http://news.com.com/` [Accessed: June 17, 2005].

[123] J. Oikarinen and D. Reed. Internet Relay Chat protocol, May 1993. RFC 1459, Status: Experimental. `http://www.faqs.org/rfcs/rfc1459.html` [Accessed: June 10, 2004].

[124] Open Source. aMSN : Alvaro's Messenger, or Another MSN Messenger clone. `http://amsn.sourceforge.net/index.php` [Accessed: May 3, 2005].

[125] Open Source. Gaim: A multi-protocol Instant Messaging (IM) client. Version 1.0.2, `http://gaim.sourceforge.net/` [Accessed: Mar. 2, 2005].

[126] Open Source. Gaim-e – encryption plug-in for gaim. `http://gaim-e.sourceforge.net/` [Accessed: Mar. 17, 2005].

[127] Open Source. The gnu privacy guard. `http://www.gnupg.org/` [Accessed: Mar. 17, 2005].

[128] Open Source. The jabberd project. Version 2.0s6, `http://jabberd.jabberstudio.org/2/` [Accessed: Mar. 2, 2005].

[129] Open Source. MySQL Database Server. Version 12.22, `http://www.mysql.com/` [Accessed: Mar. 31, 2005].

[130] Open Source. OpenSSL: The open source toolkit for SSL/TLS. Version 0.9.7e, `http://www.openssl.org/` [Accessed: Mar. 2, 2005].

[131] G. L. Orgill, G. W. Romney, M. G. Bailey, and P. M. Orgill. The urgency for effective user privacy-education to counter social engineering attacks on secure computer systems. In *Proceedings of the 5th Conference on Information Technology Education*, pages 177–181. ACM Press, 2004.

[132] V. N. Padmanabhan and L. Subramanian. An investigation of geographic mapping techniques for internet hosts. *ACM SIGCOMM Computer Communication Review*, 31(4):173–185, 2001.

[133] D. Park, C. Boyd, and S.-J. Moon. Forward secrecy and its application to future mobile communications security. In *Proceedings of the 3rd International Workshop on Practice and Theory in Public Key Cryptography (PKC 2000)*, volume 1751 of *LNCS*, pages 433–445. Springer-Verlag, Jan. 2000.

[134] R. Pastor-Satorras and A. Vespignani. Epidemic spreading in scale-free networks. *Physical Review Letters*, 86(14):3200–3203, 2001.

[135] S. Patel. Number theoretic attacks on secure password schemes. In *18th IEEE Computer Society Symposium on Research in Security and Privacy*, pages 236–247, 1997.

[136] V. Paxson, editor. *Proceedings of the 2004 ACM Workshop on Rapid Malcode (WORM'04)*, Washington, DC, USA, Oct. 2004. ACM Press.

[137] B. Pinkas and T. Sander. Securing passwords against dictionary attacks. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 161–170, Washington, DC, USA, 2002. ACM Press.

[138] B. Preneel, A. Bosselaers, and H. Dobbertin. The cryptographic hash function RIPEMD-160. *CryptoBytes*, 3(2):9–14, 1997.

[139] E. Rescorla. *SSL and TLS: Designing and Building Secure Systems*. Addison Wesley, 2001.

[140] Reuters. Reuters Messaging. `http://about.reuters.com/productinfo/messaging/` [Accessed: Mar. 22, 2005].

[141] P. Riikonen. SILC protocol white paper, Oct. 2003. `http://www.silcnet.org/docs/silc_protocol.pdf` [Accessed: Sep. 22, 2004].

[142] P. Riikonen. Secure Internet Live Conferencing (SILC), protocol specification, Feb. 2004. Internet-Draft. `http://www.silcnet.org/docs/draft-riikonen-silc-spec-08.txt` [Accessed: Sep. 22, 2004].

[143] P. Riikonen. SILC key exchange and authentication protocols, Feb. 2004. Internet-Draft. `http://www.silcnet.org/docs/draft-riikonen-silc-ke-auth-08.txt` [Accessed: Sep. 22, 2004].

[144] R. Rivest. The MD5 message-digest algorithm, Apr. 1992. RFC 1321, Status: Informational. `http://www.ietf.org/rfc/rfc1321.txt` [Accessed: Apr. 4, 2005].

[145] R. L. Rivest and A. Shamir. How to expose an eavesdropper. *Communications of the ACM*, 27(4):393–394, 1984.

[146] A. B. Roach. Session Initiation Protocol (SIP) - specific event notification, June 2002. RFC 3265, Status: Proposed Standard. `http://www.faqs.org/rfcs/rfc3265.html` [Accessed: June 22, 2004].

[147] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol, June 2002. RFC 3261, Status: Proposed Standard. `http://www.faqs.org/rfcs/rfc3261.html` [Accessed: June 22, 2004].

[148] P. Saint-Andre. Extensible messaging and presence protocol (XMPP): Core, Oct. 2004. RFC 3920, Status: Standards Track. `http://www.ietf.org/rfc/rfc3920.txt` [Accessed: Mar. 2, 2005].

[149] P. Saint-Andre. Extensible messaging and presence protocol (XMPP): Instant Messaging and presence, Oct. 2004. RFC 3921, Status: Standards Track. `http://www.ietf.org/rfc/rfc3921.txt` [Accessed: Mar. 3, 2005].

[150] SANS. Internet storm center. `http://isc.sans.org/` [Accessed: Mar. 23, 2005].

[151] S. Savage, editor. *Proceedings of the 2003 ACM Workshop on Rapid Malcode (WORM'03)*, Washington, DC, USA, Oct. 2003. ACM Press.

[152] Secunia. ICQ predictable file location weakness, Feb. 2004. Secunia Advisory SA10970, `http://secunia.com/advisories/10970/` [Accessed: June 17, 2005].

[153] A. Shamir and N. van Someren. Playing "hide and seek" with stored keys. In *Proceedings of the 3rd International Conference on Financial Cryptography*, volume 1648 of *LNCS*, pages 118–124. Springer-Verlag, 1999.

[154] M. Smith, J. J. Cadiz, and B. Burkhalter. Conversation trees and threaded chats. In *Computer Supported Cooperative Work*, pages 97–105, 2000.

[155] R. D. Smith. Instant Messaging as a scale-free network, July 2004. cond-mat/0206378, `http://arxiv.org/abs/cond-mat/0206378` [Accessed: Mar. 20, 2005].

[156] S. Staniford, V. Paxson, and N. Weaver. How to 0wn the Internet in your spare time. In *Proceedings of the 11th USENIX Security Symposium*, pages 149–167. USENIX Association, 2002.

[157] S. G. Stubblebine and P. C. van Oorschot. Addressing online dictionary attacks with login histories and humans-in-the-loop (extended abstract). In *Proceedings of the 8th International Conference on Financial Cryptography (FC 2004)*, volume 3110 of *LNCS*, pages 39–53. Springer-Verlag, 2004.

[158] K. Subramanyam, C. E. Frank, and D. H. Galli. Keyloggers: The overlooked threat to computer security. In *1st Midstates Conference for Undergraduate Research in Computer Science and Mathematics*, Oct. 2003. `http://www.denison.edu/mathsci/mcurcsm2003/papers/keyloggers.pdf` [Accessed: Mar. 7, 2004].

[159] Symantec. Norton antivirus 2005. `http://www.symantec.com/nav/nav_9xnt/` [Accessed: May 2, 2005].

[160] Symantec. Search and latest virus threats page. http://www.symantec.com/avcenter/vinfodb.html [Accessed: Mar. 18, 2005].

[161] Symantec. Securing Instant Messaging. http://securityresponse.symantec.com/avcenter/reference/secure.instant.messaging.pdf [Accessed: Jun. 27, 2004].

[162] Symantec. W32.Bizex.Worm. http://securityresponse.symantec.com/avcenter/venc/data/w32.bizex.worm.html [Accessed: June 21, 2005].

[163] Symantec. W32.Bropia.M. http://securityresponse.symantec.com/avcenter/venc/data/w32.bropia.m.html [Accessed: June 21, 2005].

[164] Symantec. W32.Choke.Worm. http://securityresponse.symantec.com/avcenter/venc/data/w32.choke.worm.html [Accessed: June 21, 2005].

[165] Symantec. W32.Serflog.A. http://securityresponse.symantec.com/avcenter/venc/data/w32.serflog.a.html [Accessed: June 21, 2005].

[166] Symantec. W95.SoFunny.Worm@m. http://securityresponse.symantec.com/avcenter/venc/data/w95.sofunny.worm@m.html [Accessed: June 21, 2005].

[167] TechWeb.com Staff. IM threats growing 50% per month, Mar. 2005. TechWeb.com news article, http://informationweek.com [Accessed: Mar. 25, 2005].

[168] Techworld.com Staff. Real Player struck by massive security hole, Feb. 2004. News article, http://www.techworld.com/news/ [Accessed: Sep. 15, 2004].

[169] The CAPTCHA Project. Telling humans and computer apart (automatically). http://www.captcha.net/ [Accessed: Mar. 19, 2005].

[170] Trend Micro. JS_MENGER.GEN. http://www.trendmicro.com/vinfo/virusencyclo/default5.asp?VName=JS_MENGER.GEN [Accessed: June 21, 2005].

[171] Trend Micro. Virus encyclopedia. http://www.trendmicro.com/vinfo/virusencyclo/ [Accessed: Mar. 18, 2005].

[172] Trend Micro. WORM_AGOBOT.AJC. http://www.trendmicro.com/vinfo/virusencyclo/default5.asp?VName=WORM_AGOBOT.AJC [Accessed: June 21, 2005].

[173] Trend Micro. WORM_BROPIA.F. http://www.trendmicro.com/vinfo/virusencyclo/default5.asp?VName=WORM_BROPIA.F [Accessed: June 21, 2005].

[174] G. Tsudik and E. V. Herreweghen. Some remarks on protecting weak keys and poorly-chosen secrets from guessing attacks. In *IEEE Symposium on Reliable Distributed Systems*, pages 136–142, 1993.

[175] US-CERT. AOL Instant Messenger contains buffer overflows in parsing of AIM URI handler requests, Jan. 2002. `http://www.kb.cert.org/vuls/id/474592` [Accessed: July 8, 2004].

[176] U.S. Securities and Exchange Commission. Sarbanes-Oxley Act, Jan. 2002. `http://www.sarbanes-oxley.com/` [Accessed: May 17, 2005].

[177] USA Today Staff. IM viruses opening a new can of worms, Aug. 2001. News article, `http://www.usatoday.com` [Accessed: Mar. 17, 2005].

[178] P. C. van Oorschot and M. J. Wiener. On Diffie-Hellman key agreement with short exponents. In *Advances in Cryptology - Eurocrypt '96*, volume 1070 of *LNCS*, pages 332–343. Springer-Verlag, 1996.

[179] Venkat. Yahoo Messenger Protocol (ver 11). `http://www.venkydude.com/articles/yahoo.htm` [Accessed: Dec. 7, 2003].

[180] J. Viega, M. Messier, and P. Chandra. *Network Security with OpenSSL*. O'Reilly, 2002.

[181] J.-M. Wams and M. van Steen. Pervasive messaging, Mar. 2003. `http://citeseer.nj.nec.com/560838.html` [Accessed: Dec. 7, 2003].

[182] S. Whittaker, D. Frohlich, and O. Daly-Jones. Informal workplace communication: What is it like and how might we support it? In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 131–137, Boston, MA, USA, 1994. ACM Press.

[183] M. Williamson. Throttling viruses: Restricting propagation to defeat malicious mobile code. In *Proceedings of the ACSAC Security Conference*, pages 61–68, Dec. 2002.

[184] M. Williamson and A. Parry. Virus throttling for Instant Messaging. In *Virus Bulletin Conference, vb2004*, Sept. 2004.

[185] T. Wu. The secure remote password protocol. In *Proceedings of the Internet Society Network and Distributed System Security Symposium*, pages 97–111, Mar. 1998.

[186] T. Wu. The SRP authentication and key exchange system, Sept. 2000. RFC 2945, Status: Standards Track. `http://www.faqs.org/rfcs/rfc2945.html` [Accessed: Sep. 28, 2004].

[187] Yahoo! Inc. Yahoo! Business Messenger. `http://messenger.yahoo.com/messenger/business/` [Accessed: Mar. 22, 2005].

[188] Yahoo! Inc. Yahoo! Messenger. `http://messenger.yahoo.com/` [Accessed: Mar. 22, 2005].

[189] M. Zhang. Analysis of the SPEKE password-authenticated key exchange protocol. *IEEE Communications Letters*, 8:63–65, Jan. 2004.

[190] M. Zhang. New approaches to password authenticated key exchange based on RSA. In *Advances in Cryptology - Asiacrypt 2004*, LNCS, pages 230–245. Springer-Verlag, Dec. 2004.

[191] S. W. Zhiguo Wan. Cryptanalysis of two password-authenticated key exchange protocols. In *Proceedings of the Information Security and Privacy: 9th Australasian Conference (ACISP 2004)*, volume 3108 of *LNCS*, pages 164–175. Springer-Verlag, July 2004.

[192] Zone Labs. IMSecure. `http://www.zonelabs.com/` [Accessed: Mar. 17, 2005].

[193] Zone Labs. ZoneAlarm. `http://www.zonelabs.com/` [Accessed: Mar. 17, 2005].